Creo® Parametric TOOLKIT User's Guide

8.0.2.0

PTC Inc., 121 Seaport Blvd, Boston, MA 02210 USA

Contents

About This Guide

Creo Parametric TOOLKIT is the C-language customization toolkit for Creo Parametric. It provides customers and third-parties the ability to expand Creo Parametric capabilities by writing C-language code and seamlessly integrating the resulting application into Creo Parametric. It provides a large library of C functions that enables the external application to access the Creo Parametric database and user interface in a controlled and safe manner.

Note

Creo Parametric TOOLKIT is supported only with Creo Parametric. It is not supported with the other Creo applications.

Creo Parametric TOOLKIT is aimed at software engineers with experience in C programming. They should also be trained in the basic use of Creo Parametric.

The *Creo Parametric TOOLKIT User's Guide* describes how to use Creo Parametric TOOLKIT. This manual introduces Creo Parametric TOOLKIT, the features it offers, and the techniques and background knowledge users require to use it.

Note

The code examples included in this guide have been reformatted for presentation purposes, and therefore may contain hidden editing characters, such as tabs and end-of-line characters, and extraneous spaces. Do not cut and paste sample application code or code fragments from the User's Guide as the additional formatting characters could break the code; always use the sample code provided in the Creo Parametric TOOLKIT installation directory at `<creo_loadpoint>\<datecode>\Common Files\protoolkit\protk_appls`.

The documentation for Creo Parametric TOOLKIT includes an online browser that contains the *Creo Parametric TOOLKIT User's Guide* and describes Creo Parametric TOOLKIT function syntax.

The following table lists conventions and terms used throughout this book.

Convention	Description
UPPERCASE	Creo Parametric-type menu name (for example, PART).
Boldface	Windows-type menu name or menu or dialog box option (for example, **View**), or utility (for example, **promonitor**). Function names also appear in boldface font.
`Monospace(Courier)`	Code samples appear
Emphasis	Important information appears in italics. Italic font also indicates file names and function arguments.
Choose	Highlight a menu option by placing the arrow cursor on the option and pressing the left mouse button.
Select	A synonym for "choose" as above, Select also describes the actions of selecting elements on a model and checking boxes.
Element	An element describes redefinable characteristics of a feature in a model.
Mode	An environment in Creo Parametric in which you can perform a group of closely related functions (Drawing, for example).
Model	An assembly, part, drawing, format, notebook, case study, sketch, and so on.
Option	An item in a menu or an entry in a configuration file or a setup file.
Solid	A part or an assembly.

Convention	Description
<creo_loadpoint>	The location where the Creo applications are installed, for example, C:\Program Files\PTC\Creo 1.0.
<creo_toolkit_loadpoint>	The location where the Creo Parametric TOOLKIT application is installed, that is, <creo_loadpoint>\<datecode>\Common Files\protoolkit.

Note

- Important information that should not be overlooked appears in notes like this.
- All references to mouse clicks assume use of a right-handed mouse.

Software Product Concerns and Documentation Comments

For resources and services to help you with PTC software products, please visit the Customer Support Guide in the eSupport Portal at:

http://support.ptc.com/appserver/support/csguide/csguide.jsp

You must have a Service Contract Number (SCN) before you can receive assisted technical support. If you do not have a service contract number please contact PTC Customer Care by clicking the Contact tab on the Customer Support Guide page.

In regard to documentation, PTC welcomes your suggestions and comments. You can send feedback in the following ways:

- Send comments electronically to *MCAD-documentation@ptc.com*.
- Fill out and mail the PTC Documentation Survey in the customer service guide.

1

Fundamentals

This chapter describes fundamental Creo Parametric TOOLKIT concepts and functions.

Introduction to Creo Parametric TOOLKIT

Creo Parametric TOOLKIT is the customization toolkit for Creo Parametric from Parametric Technology Corporation (PTC). It gives customers and third-parties the ability to expand Creo Parametric capabilities by writing C programming language code and then seamlessly integrating the resulting application into Creo Parametric.

Creo Parametric TOOLKIT provides a large library of C functions to provide the external application safe and controlled access to the Creo Parametric database and applications. Creo Parametric TOOLKIT is the primary PTC application programmer's interface (API) for Creo Parametric.

Online Documentation in Creo Parametric TOOLKIT APIWizard

Creo Parametric TOOLKIT provides an online browser called the APIWizard that displays detailed documentation data. This browser displays information from the *Creo Parametric TOOLKIT Users' Guide* and API specifications derived from Creo Parametric TOOLKIT header file data.

The Creo Parametric TOOLKIT APIWizard contains the following:

- Definitions of Creo Parametric TOOLKIT objects and their hierarchical relationships
- Definitions of Creo Parametric TOOLKIT functions
- Declarations of data types used by Creo Parametric TOOLKIT functions
- The *Creo Parametric TOOLKIT Users' Guide*, which users can browse by topic or by object
- Code examples for Creo Parametric TOOLKIT functions (taken from applications provided as part of the Creo Parametric TOOLKIT installation)

Review the Release Notes for the most up-to-date information on documentation changes.

Note

- The User's Guide is also available in PDF format. This file is located at:`<creo_toolkit_loadpoint>\tkuse.pdf`
- From Creo Parametric 4.0 F000, the applet based APIWizard is no longer supported. Use the non-applet based APIWizard instead.

To Install the APIWizard

The Creo Parametric product CD installation procedure automatically installs the Creo Parametric TOOLKIT APIWizard. The files reside in a directory under the Creo Parametric load point. The location for the Creo Parametric TOOLKIT APIWizard files is `<creo_toolkit_loadpoint>\protkdoc`

To load the APIWizard manually, copy all files from `<creo_toolkit_loadpoint>\protkdoc` to your target directory.

APIWizard Overview

The APIWizard supports Internet Explorer, Firefox, and Chromium browsers.

Start the Creo Parametric TOOLKIT APIWizard by pointing your browser to: `<creo_toolkit_loadpoint>\protkdoc\index.html`

A page containing links to the Creo Parametric TOOLKIT APIWizard and User's Guide will open in the web browser.

APIWizard

Click APIWizard to open the list of Creo Parametric TOOLKIT Objects and the related functions. Click a function name to read more about it.

Use the search field at the top of the left pane to search for functions. You can search using the following criteria:

- Search by API names
- Search using wildcard character *, where * (asterisk) matches zero or more nonwhite space characters

The search displays the resulting API names with embedded links in a drop down list. The deprecated APIs are highlighted in yellow.

User's Guide

Click User's Guide to access the *Creo Parametric Toolkit User's Guide*.

Creo Parametric TOOLKIT Style

Creo Parametric TOOLKIT uses an object-oriented programming style. Data structures for the transfer information between Creo Parametric and the application are not directly visible to the application. These data structures are accessible only with Creo Parametric TOOLKIT functions.

Objects and Actions

The most basic Creo Parametric TOOLKIT concepts are objects and actions.

Each Creo Parametric TOOLKIT library C function performs an action on a specific type of object. The Creo Parametric TOOLKIT convention for function names is the prefix “Pro” the name of the object type, and the name of the action it performs, for example:

```
ProSectionLocationGet()
```

A Creo Parametric TOOLKIT object is a well-defined and self-contained C structure used to perform actions relevant to that object. Most objects are items in the Creo Parametric database, such as features and surfaces. Others, however, are more abstract or transient, such as the information resulting from a select action.

In Creo Parametric TOOLKIT, each type of object has a standard name consisting of a “Pro” plus capitalized word that describes the object. Simple examples of Creo Parametric TOOLKIT object types and their Creo Parametric equivalents are as follows:

- `ProFeature`—A feature
- `ProSurface`—A surface
- `ProSolid`—An abstract object created to exploit the commonality between parts and assemblies
- `ProWcell`—A workcell feature in a manufacturing assembly

Creo Parametric TOOLKIT provides a C typedef for each object used for variables and arguments that refer to those objects. Creo Parametric TOOLKIT objects have a hierarchical relationship that reflects the Creo Parametric database. For example, a `ProFeature` object can contain objects of type `ProSurface` (among others).

For example, the following functions have actions that are single verbs:

```
ProSolidRegenerate()
```

```
ProFeatureDelete()
```

Some Creo Parametric TOOLKIT functions require names that include more than one object type. The function names have the object types first, then the action. For example:

```
ProFeatureParentsGet()
```

```
ProWcellTypeGet()
```

The action verbs indicate the type of action being performed, as shown in the following table.

Action Verb	Type of Action
Get	Read information directly from the Creo Parametric database.
Eval	Provide the result of a simple calculation.
Compute	Provide the result of a computation that typically involves numerical analysis of the model geometry.

Examples are:

- `ProEdgeLengthEval()`
- `ProSurfaceAreaEval()`
- `ProSolidRayIntersectionCompute()`

To illustrate further, function `ProSolidOutlineGet()` reads from Creo Parametric the currently stored solid outline, but `ProSolidOutlineCompute()` invokes a recomputation of that information. Use `ProSolidOutlineCompute()` to compute an accurate outline of a solid.

Note

Do not use `ProSolidOutlineGet()` to calculate the outline of a solid. It will not return a properly calculated outline.

Other Creo Parametric TOOLKIT function conventions are that the first argument identifies the object, and input arguments come before output arguments.

Function Prototyping

Each Creo Parametric TOOLKIT function has an ANSI function prototype. (The C compilers on platforms supported by Creo Parametric TOOLKIT provide at least the option of function prototype checking.) All function prototypes for a particular Creo Parametric TOOLKIT object reside in a header file named for that object. For example, the prototype for function `ProEdgeLengthEval()` is located in the header file `ProEdge.h`.

Note

PTC strongly recommends that you use prototyping. Make sure you include the appropriate header files in your Creo Parametric TOOLKIT application.

Function Error Statuses

The return type of most Creo Parametric TOOLKIT functions is `ProError`. `ProError` is an enumerated type with a value for each common case where Creo Parametric TOOLKIT functions succeeds or fails.

The normal value for success is `PRO_TK_NO_ERROR`. The other "failure" statuses occur when there is a genuine problem, or for more benign reasons. For example, these error statuses denote genuine problems:

- `PRO_TK_BAD_INPUTS`—The Creo Parametric TOOLKIT program called the function incorrectly.
- `PRO_TK_OUT_OF_MEMORY` or `PRO_TK_COMM_ERROR`—System failure.

The following statuses are more benign:

- `PRO_TK_USER_ABORT`—A function that supports user interaction was aborted by the Creo Parametric user.
- `PRO_TK_E_NOT_FOUND`—A function attempted operation on an empty object list.

Users must pay careful attention to how their program reacts to a Creo Parametric TOOLKIT function error status—there can be several types of failure and success, each requiring different handling.

The subset of `ProError` values that a particular Creo Parametric TOOLKIT function can return is described in the browser under that function. Possible errors are also included in a comment under each function prototype in the corresponding Creo Parametric TOOLKIT header file.

Installing Creo Parametric TOOLKIT

The next sections describe how to install Creo Parametric TOOLKIT.

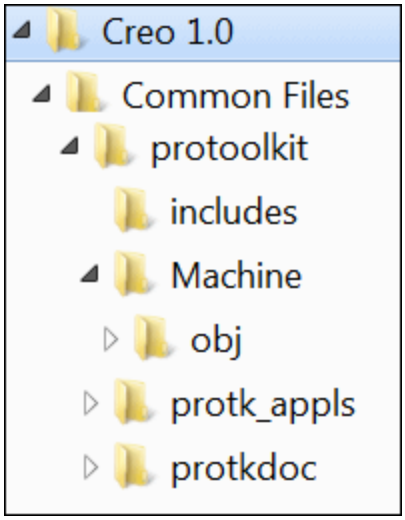
Overview

Creo Parametric TOOLKIT is on the same DVD as Creo Parametric, so you do not need to arrange a special delivery from your supplier. When Creo Parametric is installed using PTC.Setup, one of the optional components is `API Toolkits`. This includes Creo Parametric TOOLKIT, Pro/WebLink, J-Link, VB, and Creo Object TOOLKIT C++ and Creo Object TOOLKIT Java.

If you select Creo Parametric TOOLKIT, it is installed under the loadpoint of Creo Parametric at the location `<creo_loadpoint>\<datecode>\Common Files\protoolkit`. The `protoolkit` directory contains all the headers, libraries, example applications, and documentation specific to Creo Parametric TOOLKIT.

The following figure shows the tree of directories found under the Creo Parametric TOOLKIT loadpoint after installation.

Creo Parametric TOOLKIT Installation Directory Tree



The directory `protk_appls` contains sample Creo Parametric TOOLKIT applications. For more information regarding the sample applications refer to the Appendix on Sample Applications.

Add or Update Creo Parametric TOOLKIT Installation

Add a Creo Parametric TOOLKIT installation to an existing Creo Parametric installation using the Update option in PTC.Setup. For a description of using PTC.Setup, refer to the *Creo Parametric Installation and Administration Guide*.

Be sure your system administrator reinstalls Creo Parametric TOOLKIT each time they update your Creo Parametric installation from a new CD. PTC recommends that, when possible, you use a Creo Parametric TOOLKIT from the same build number as Creo Parametric.

Note

The Creo Parametric library functions work by invoking functions inside the Creo Parametric executable, so an update to Creo Parametric TOOLKIT often involves a change to Creo Parametric rather than Creo Parametric TOOLKIT itself. So when you receive a Creo Parametric DVD that contains an update to Creo Parametric TOOLKIT, always reinstall Creo Parametric from that DVD.

In many situations it will be inconvenient or impossible to ensure that the users of your Creo Parametric TOOLKIT application will use the same build of Creo Parametric that you used to compile and link the Creo Parametric TOOLKIT application. Refer to section Version Compatibility: Creo Parametric and Creo Parametric TOOLKIT on page 41 for the rules to mix versions of Creo Parametric and Creo Parametric TOOLKIT.

Testing the Creo Parametric TOOLKIT Installation

After your system administrator has installed Creo Parametric TOOLKIT, you should compile, link, and run a simple Creo Parametric TOOLKIT application as soon as possible on each machine you intend to use for development. This provides an independent test of the following items:

- The installation of Creo Parametric TOOLKIT is present, complete, and visible from your machine.
- The version of Creo Parametric you plan to use during development has the Creo Parametric TOOLKIT license option added to it.
- The machine you will use for development has access to all the necessary C program development tools, in versions supported by Creo Parametric TOOLKIT (especially, of course, the C compiler and linker).

Running the Microsoft Visual Studio Solution

PTC provides a ready-to-use Visual Studio solution on the Windows platform to build and test Creo Parametric TOOLKIT applications by using an appropriate makefile. For the version of Visual Studio compatible with the release of Creo Parametric TOOLKIT, refer to the hardware notes at Creo Future Platform Support Summary.

This ready-to-use Visual Studio solution has the following advantages:

- Provides an effective way to build and test sample applications provided by PTC.
- Provides a preconfigured Visual Studio development environment for use with Creo Parametric TOOLKIT.
- Supports `Intellisense` for Creo Parametric TOOLKIT functions.

Note

- The supported version of Visual Studio changes with every release of Creo Parametric TOOLKIT, and hence the compiler flags and libraries also change. For every release, you must download the latest version of the ready-to-use Visual Studio solution from the `creo_toolkit_loadpoint`.
- In Creo Parametric 7.0.1.0 and later, Creo Parametric TOOLKIT supports Visual Studio 2019. The compiler flags and libraries are available for Visual Studio 2019. Creo Parametric TOOLKIT no longer supports Visual Studio 2015.

  All Creo Parametric TOOLKIT applications on 64-bit Windows platforms built using the Microsoft Visual Studio 2019 compiler must set the configuration property Platform Toolset as Visual Studio 2019 (v142).

When you install Creo Parametric TOOLKIT, the file `protk_install_example.zip` is installed under the `<creo_toolkit_loadpoint>` at `$<machine_type>\obj`. To use this solution:

1. Unzip `protk_install_example.zip`. The following files and directories are available:

Directory or File	Description
`make_install.sln`	Specifies the ready-to-use Visual Studio solution file.
`make_install`	Contains the makefile project and the `creotk.dat` file.

2. Open Microsoft Visual Studio.
3. Click **File ► Open ► Project/Solution**. The **Open Project** dialog opens.
4. Browse the `protk_install_example` directory and select `make_install.sln`.
5. Click **Open** to access the solution file.

The `make_install` makefile project is available in Visual Studio.

Running the Makefile Project

1. Click **Build ► Build make_install**. The application should build without errors. This creates the Creo Parametric TOOLKIT DLL file called `pt_inst_`

`test.dll`. If the application fails, check that the environment variable `PROTOOL_SRC` is set correctly.

2. Modify the `exec_file` and `text_dir` fields in the `creotk.dat` file located in the `make_install` directory to specify the full path to `pt_inst_test.dll` and `\text`, respectively. For example,

```
exec_file <full_path>\pt_inst_test.dll
text_dir <full_path>\text
```

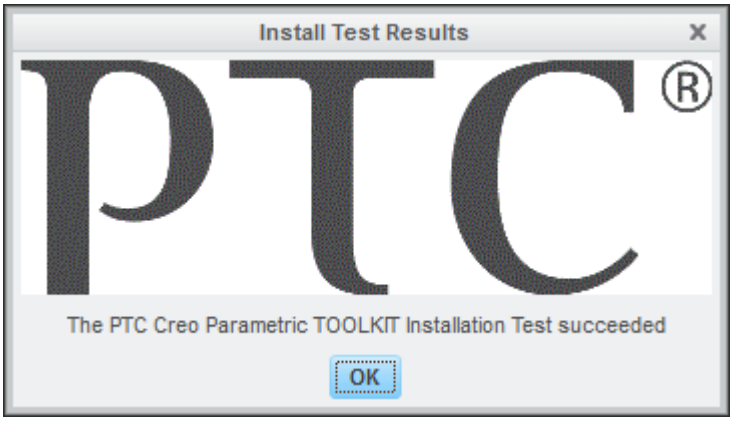
3. Unlock the application, using the following command:

```
<creo_loadpoint>\<datecode>\Parametric\bin\protk_unlock.bat
<path to executables or DLLs to unlock>
```

4. Start Creo Parametric.
5. On the **Tools** tab, in the **Utilities** group, click **Auxiliary Applications**. The **Auxiliary Applications** dialog box opens.
6. Click **Register** to register the updated `creotk.dat` file. The **Register auxiliary application** dialog box opens.
7. Browse to the `<full_path>` and select `creotk.dat`.
8. Click **Open**. The Creo Parametric TOOLKIT application adds the command **Install Test** under the **Tools ► File** menu in the **TOOLKIT** group in the **Home** tab on the Creo Parametric ribbon user interface.

> **Note**
>
> Refer to the Creo Parametric Help for more information on customizing the Ribbon.

9. Click **Tools** and then click **File ► Install Test**. The Creo Parametric TOOLKIT **Install Test Results** message window opens, indicating that the installation test has succeeded.



10. Click **OK**.

To run other sample applications provided by PTC, follow these steps:

1. Copy the required makefile from <creo_toolkit_loadpoint>\$<machine_type>\obj to the make_install directory of the ready-to-use Visual Studio solution.

   If you are working on a 64-bit Windows platform, copy the file from <creo_toolkit_loadpoint>\x86e_win64\obj.
2. Copy the text directory associated with the sample application from <creo_toolkit_loadpoint>\protk_appls\<app_name>\text to the make_install directory.
3. Open Visual Studio and set the values of the following variables in the makefile:
   ```
   PROTOOL_SRC = ../../../../../protoolkit
   PROTOOL_SYS = $(PROTOOL_SRC)/$(PRO_MACHINE_TYPE)
   ```
4. Click **Project ▸ Properties** to update the **NMake** properties of the project.
5. Click **Build ▸ Rebuild make_install**. The application builds and creates a new .dll file.
6. Update the creotk.dat file located in the make_install directory with the name of the sample application and the DLL file.
7. Modify the exec_file and text_dir fields in the creotk.dat file to specify the full path to the .dll file and \text directory, respectively.
8. Start Creo Parametric.
9. On the **Home** tab, in the **Utilities** group, click **Auxiliary Applications** or click **Tools ▸ Auxiliary Applications**. The **Auxiliary Applications** dialog box opens.
10. Click **Register** to register the updated creotk.dat file. The **Register auxiliary application** dialog box opens.
11. Browse to the full path and select creotk.dat.
12. Click **Open**. The Creo Parametric TOOLKIT application runs.

Building a Sample Application

The Creo Parametric TOOLKIT loadpoint includes the source of a simple application designed specifically to test the Creo Parametric TOOLKIT installation. The steps required to build and run the test application are described in the following sections.

In this explanation, <creo_toolkit_loadpoint> refers to the directory that forms the loadpoint of Creo Parametric TOOLKIT, and <machine_type> refers to the name of the type of platform you are using (for example, i486_nt).

Step 1—Compile and Link

Compile and link the Creo Parametric TOOLKIT installation test application using the makefile `<creo_toolkit_loadpoint>\$<machine_type>\obj\make_install`

The makefile is designed to be run in that location, and creates a Creo Parametric TOOLKIT application file in the directory from which it is run. If you do not have root privileges, you probably need to copy the makefile to a directory of your own so the output file can be created. If you do this, you also need to edit the makefile to correct the macro that refers to the Creo Parametric TOOLKIT loadpoint.

If you copy the makefile to another directory, replace the line:

```
PROTOOL_SRC = ../..
```

with:

```
PROTOOL_SRC = <creo_toolkit_loadpoint>
```

In this line, `<creo_toolkit_loadpoint>` is the loadpoint for your Creo Parametric TOOLKIT installation.

To run the makefile, type the following command:

```
nmake -f make_install
```

This creates a file called `pt_inst_test.exe`.

If you experience any error messages at this stage, it might be due to the Creo Parametric TOOLKIT installation being incomplete, or, more likely, the C compiler and linker being unavailable or unsupported by Creo Parametric TOOLKIT.

Step 2—Register

In the same directory, create a text file called `creotk.dat`. This file is the "registry file" that tells Creo Parametric about the Creo Parametric TOOLKIT application. Refer to the Registering a Creo Parametric TOOLKIT Application on page 38 and Sample Registry Files on page 2101 sections for syntax requirements for this file. The `creotk.dat` file should contain the following lines:

```
name install_test
exec_file pt_inst_test.exe
text_dir <creo_toolkit_loadpoint>/protk_appls/pt_install_test
end
```

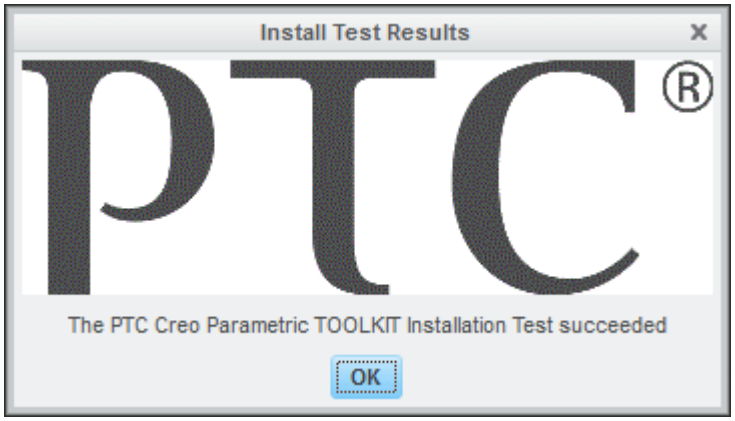
Note

Use the delimiter character `\` in `creotk.dat`.

Step 3—Run Creo Parametric

Run Creo Parametric from the directory that contains the `creotk.dat` file; Creo Parametric starts the Creo Parametric TOOLKIT application in multiprocess mode (see the section How Creo Parametric TOOLKIT Works on page 35 for more information on multiprocess mode). You should see that the **Install Test** command has been added in the **TOOLKIT** group in the **Home** tab on the Creo Parametric ribbon user interface. Click **Tools** and then click **File ▸ Install Test**. The Creo Parametric TOOLKIT application displays a custom dialog indicating whether the installation test has succeeded:

Install Test Results Dialog Box



Failure or error messages at this stage could be due to the following reasons:

- You made a mistake when creating the `creotk.dat` file. If the syntax or contents are wrong, you should see a self-explanatory message in the window from which you started Creo Parametric.
- The Creo Parametric you ran is not licensed for Creo Parametric TOOLKIT. This also causes an explanatory message to be displayed in the startup window.
- The Creo Parametric TOOLKIT executable you created in Step 1 is wrong in some way: it is for the wrong platform, for example, or might not have execute access. You can check this by trying to execute the file directly by typing its name. If the file is correct, the program prints the following messages and then terminates:

```
pt_inst_test: insufficient arguments; need 2 arguments:
   (1) own RPC program #
   (2) root directory path for Pro/TOOLKIT text files.
```

If the file is incorrect, the exact message will depend on which platform you are using, but should explain the cause of the problem.

Step 4—Repeat the Test in DLL Mode

To build for DLL mode, use the same makefile, but use the following line instead of the line `nmake -f make_install`:

```
nmake -f make_install dll
```

This creates a file called `pt_inst_test.dll`, which is the library to be dynamically linked.

Next, make these two changes to the `creotk.dat` file:

Add this line after the first line:

```
startup dll
```

Change the `exec_file` statement to reference the new Creo Parametric TOOLKIT file. Use the executable name `pt_inst_test.exe` in the second line, and the Windows directory syntax in the third line.

You can run Creo Parametric and look at the behavior of the Creo Parametric TOOLKIT application exactly as in Step 3.

See the section How Creo Parametric TOOLKIT Works on page 35 for more information on DLL mode.

Developing a Creo Parametric TOOLKIT Application

This section describes how Creo Parametric TOOLKIT works and the steps you need to take after installing Creo Parametric TOOLKIT to create a Creo Parametric TOOLKIT application. The topics are as follows:

How Creo Parametric TOOLKIT Works

The standard method by which Creo Parametric TOOLKIT application code is integrated into Creo Parametric is through the use of dynamically linked libraries (DLLs). When you compile your Creo Parametric TOOLKIT application C code

and link it with the Creo Parametric TOOLKIT library, you create an object library file designed to be linked into the Creo Parametric executable when Creo Parametric starts up. This method is referred to as `DLL mode`.

Creo Parametric TOOLKIT also supports a second method of integration: the "multiprocess," or spawned mode. In this mode, the Creo Parametric TOOLKIT application code is compiled and linked to form a separate executable. This executable is designed to be spawned by Creo Parametric and runs as a child process of the Creo Parametric session. In DLL mode, the exchanges between the Creo Parametric TOOLKIT application and Creo Parametric are made through direct function calls. In multiprocess mode, the same effect is created by an inter-process messaging system that simulates direct function calls by passing the information necessary to identify the function and its argument values between the two processes.

Multiprocess mode involves more communications overhead than DLL mode, especially when the Creo Parametric TOOLKIT application makes frequent calls to Creo Parametric TOOLKIT library functions, because of the more complex method of implementing those calls. However, it offers the following advantage: it enables you to run the Creo Parametric TOOLKIT application with a source-code debugger without also loading the whole Creo Parametric executable into the debugger. See the section Using a Source-Code Debugger on a Creo Parametric TOOLKIT Application on page 44 for more details.

You can use a Creo Parametric TOOLKIT application in either DLL mode or multiprocess mode without changing any of the C source code in the application. (The methods of setting the mode are described in detail later in this chapter.)

It is also possible to use more than one Creo Parametric TOOLKIT application within a single session of Creo Parametric, and these can use any combination of modes.

If you use multiprocess mode during development of your application to debug more easily, you should switch to DLL mode when you install the application for your end users because the performance is better in that mode. However, take care to test your application thoroughly in DLL mode before you deliver it. Any programming errors in your application that cause corruption to memory used by Creo Parametric or Creo Parametric TOOLKIT are likely to show quite different symptoms in each mode, so new bugs may emerge when you switch to DLL mode.

Although multiprocess mode involves two processes running in parallel, these processes do not provide genuine parallel processing. There is, however, another mode of integrating a Creo Parametric TOOLKIT application that provides this ability, called "asynchronous mode." (Asynchronous mode is described in detail in the chapter Core: Asynchronous Mode on page 277.) The DLL and multiprocess modes are given the general name "synchronous mode." An asynchronous Creo Parametric TOOLKIT application is fundamentally different in its architecture from a synchronous mode application, so you should choose between these

methods before writing any application code. As a general rule, synchronous mode should be the default choice unless there is some unavoidable reason to use asynchronous mode, because the latter mode is more complex to use.

Note

- All Creo Parametric TOOLKIT calls running in either synchronous (DLL or multiprocess) mode or asynchronous mode always clear the `Undo/Redo` stack in the Creo Parametric session. The Creo Parametric user interface reflects this by making the **Undo** and **Redo** menu options unavailable.
- When you invoke the Creo Parametric TOOLKIT application, ensure that no dialogs are open in Creo Parametric session. If Creo Parametric dialog is open, the results may be unpredictable.

Compiling and Linking a Creo Parametric TOOLKIT Application

This section describes compiling and linking Creo Parametric TOOLKIT applications.

Makefiles

The C compiler options and system libraries needed to compile and link a Creo Parametric TOOLKIT application are different on each platform. To ensure that the makefile you use for building your Creo Parametric TOOLKIT application uses the correct options, you should base your makefile on one of the makefiles located under the `<creo_toolkit_loadpoint>`. These are designed for building the various Creo Parametric TOOLKIT applications whose source is included in the Creo Parametric TOOLKIT installation.

An example of one of the Creo Parametric TOOLKIT applications provided is the installation test, whose source code is under the directory `<creo_toolkit_loadpoint>\protk_appls\pt_install_test`, where `<creo_toolkit_loadpoint>` is the loadpoint directory of the Creo Parametric TOOLKIT installation. The makefile for the installation test application is `<creo_toolkit_loadpoint>\$<machine_type>\obj\make_install`.

To use this as the model for your own makefile, copy it to the directory that will contain your Creo Parametric TOOLKIT source code, then make the following changes to it:

- Change the macro `MAKEFILENAME` to refer to the makefile by its new name.
- Change the macros `EXE` and `EXE_DLL` to define output file names more suitable for your own application.
- Change the macro `PROTOOL_SRC` to refer to the loadpoint of Creo Parametric TOOLKIT.
- Change the macro `OBJS` to refer to the object files that will result from compiling your Creo Parametric TOOLKIT source files.
- Add targets for those object files. These contain instructions for
- compiling your C source files. The form of these target definitions can be copied from the ones in the original makefile. They generally take the following form:

```
myfile.o:  myfile.c
        $(CC) $(CFLAGS) myfile.c
```

Note

The second line must start with a tab character.

If you want to use a debugger with your Creo Parametric TOOLKIT application, you can also add the appropriate compiler switch (usually "`-g`") to the `CCFLAGS` macro.

If you are rebuilding an existing Pro/TOOLKIT application with a new version of Creo Parametric TOOLKIT, remember to repeat these steps to set up a new makefile—do not continue to use a makefile created for the previous version. You must do this in case the compiler switches or system libraries needed to build a Creo Parametric TOOLKIT application have changed in the new version.

Registering a Creo Parametric TOOLKIT Application

Registering a Creo Parametric TOOLKIT application means providing Creo Parametric with information about the files that form the Creo Parametric TOOLKIT application. To do this, create a small text file called the Creo Parametric TOOLKIT registry file, that Creo Parametric will find and read.

Creo Parametric searches for the registry file as follows:

- In the absolute path specified in the `creotkdat`, `protkdat`, `prodevdat`, and `toolkit_registry_file` statements in the Creo Parametric configuration file.
- For the files named `creotk.dat`, `protk.dat`, or `prodev.dat` in the following locations:

1. The starting directory
2. `<creo_loadpoint>\<datecode>\Common Files\$<machine_type>\text\<language>`
3. `<creo_loadpoint>\<datecode>\Common Files\text`
4. `<creo_loadpoint>\<datecode>\Common Files\text\<language>`

In the preceding locations, the variables are as follows:

- `<creo_loadpoint>`—The Creo Parametric loadpoint (not the Creo Parametric TOOLKIT loadpoint).
- `<machine_type>`—The machine-specific subdirectory, such as, `i486_nt` or `x86e_win64`. Set the environment variable `PRO_MACHINE_TYPE` to define the type of machine on which Creo Parametric is installed.

If more than one registry file having the same filename exists in this search path, Creo Parametric stops searching after finding the first instance of the file and starts all the Creo Parametric TOOLKIT applications specified in it. If more than one registry file having different filenames exist in this search path, Creo Parametric stops searching after finding one instance of each of them and starts all the Creo Parametric TOOLKIT applications specified in them.

Note

- Option 1 is normally used during development, because the Creo Parametric TOOLKIT application is seen only if you start Creo Parametric from the specific directory that contains the registry file.
- Option 3 is recommended when making an end-user installation, because it makes sure that the registry file is found no matter what directory is used to start Creo Parametric.

The registry file is a simple text file, where each line consists of one of a predefined set of keywords, followed by a value.

The standard form of the registry file in DLL mode is as follows:

```
name YourApplicationName
startup dll
exec_file $LOADDIR/$MACHINE_TYPE/obj/filename.dll
text_dir $LOADDIR
end
```

The fields of the registry file are as follows:

- `name`—Assigns a unique name to this Creo Parametric TOOLKIT application.
- `startup`—Specifies the method Creo Parametric should use to communicate with the Creo Parametric TOOLKIT application. The example above specifies the DLL mode.
- `exec_file`—Specifies the full path and name of the file produced by compiling and linking the Creo Parametric TOOLKIT application. The example above shows a typical use of environment variables to make the reference to the executable file more flexible.
- `text_dir`—Specifies the full path name to text directory that contains the language-specific directories. The language-specific directories contain the message files, menu files, resource files and UI bitmaps in the language supported by the Creo Parametric TOOLKIT application.

Note

The fields `exec_file` and `text_dir` have a character limit of `PRO_PATH_SIZE-1` wide characters (`wchar_t`).

- `end`—Indicates the end of the description of this Creo Parametric TOOLKIT application.

If you want to run the application in multiprocess mode, make the following changes to the registry file:

- Change the `startup` statement to:
  ```
  startup spawn
  ```
- Make the `exec_file` statement refer to the Creo Parametric TOOLKIT program executable.

Note

For all Creo Parametric TOOLKIT plugins, the registry file `protk.dat` is located inside the `%ProgramData%\PTC\<Creo Parametric Toolkit version>\Plugins` subdirectories. All registry files located in this location must use the absolute path in all their entries.

For more information about the registry file, refer to the appendix Creo Parametric TOOLKIT Registry File on page 2099.

Limit on the Number of Loaded Applications

Previous versions of Pro/ENGINEER limited the number of applications that could be specified in the registry files; there is no such limit for Pro/ENGINEER Wildfire 2.0 onwards. However, most platforms do have limits for the size of a process, the total size of all processes, and the number of processes that a single process can spawn. PTC recommends that you combine related applications into the same binary file wherever possible to avoid running into these limits.

Version Compatibility: Creo Parametric and Creo Parametric TOOLKIT

In many situations it will be inconvenient or impossible to ensure that the users of your Creo Parametric TOOLKIT application use the same build of Creo Parametric used to compile and link the Creo Parametric TOOLKIT application. This section summarizes the rules for mixing Creo Parametric TOOLKIT and Creo Parametric versions. The Creo Parametric TOOLKIT version is the Creo Parametric CD version from which the user installed the Creo Parametric TOOLKIT version used to compile and link the application.

Functions Introduced:

- **ProToolkitMajorVersionGet()**

Superseded Functions

- **ProEngineerReleaseNumericversionGet()**

The function `ProToolkitMajorVersionGet()` returns the version number of the Creo Parametric executable to which the Creo Parametric TOOLKIT application is connected. This number is an absolute number and represents the major release of the product. The version number of Creo Parametric 8.0 is 40.

The function `ProEngineerReleaseNumericversionGet()` is deprecated. Use the function `ProToolkitMajorVersionGet()` instead.

Note

From Pro/ENGINEER Wildfire 4.0 onwards applications built with libraries older than Pro/ENGINEER 2001 will not run. You must recompile these applications with later versions of the Pro/TOOLKIT libraries.

The following points summarize the rules for mixing Creo Elements/Pro TOOLKIT and Creo Elements/Pro versions.

- Pro/ENGINEER release older than Pro/TOOLKIT release:

Not supported

- Creo Parametric release newer than a Creo Parametric TOOLKIT release:

  This works in many, but not all, cases. The communication method used to link Creo Parametric TOOLKIT to Creo Parametric provides full compatibility between releases. However, there are occasional cases where changes internal to Creo Parametric may require changes to the source code of a Creo Parametric TOOLKIT application in order that it continue to work correctly. Whether you need to convert Creo Parametric TOOLKIT applications depends on what functionality it uses and what functionality changed in Creo Parametric and Creo Parametric TOOLKIT. PTC makes every effort to keep these effects to a minimum. The Release Notes for Creo Parametric TOOLKIT detail any conversion work that could be necessary for that release.

- Creo Parametric build newer than Creo Parametric TOOLKIT build

  This is always supported.

Application Compatibility: Creo Parametric and Creo Parametric TOOLKIT on Different Architecture

In some situations it will be inconvenient or impossible to ensure that the users of your Creo Parametric TOOLKIT application use a machine with the same operating system and architecture as the machine on which it was compiled. An example might be an application integrating with a third party library which is only available as 32-bit architecture, but needs to be run with Creo Parametric on a 64-bit architecture machine with the same operating system. Creo Parametric TOOLKIT provides limited capability to support these situations in spawn and asynchronous mode only. DLL applications must always be compiled on machines with the same operating system and architecture as the Creo Parametric executable.

The following situations might occur:

- Creo Parametric TOOLKIT application compiled on the same architecture and operating system as Creo Parametric. This is always supported.
- Creo Parametric TOOLKIT application compiled on a machine with a smaller pointer size (native data size) than the machine on which the application is run For example, a Creo Parametric TOOLKIT application built on Windows 32-bit running on an Windows-64 bit installation of Creo Parametric. This is supported for spawn and asynchronous mode only.
- Creo Parametric TOOLKIT application compiled on a machine with a larger pointer size (native data size) than the machine on which the application is run. For example, a Creo Parametric TOOLKIT application built on Windows-

64 bit machine running on an Windows-32 bit installation of Creo Parametric. This is not supported.

Stopping and Restarting a Creo Parametric TOOLKIT Application

Creo Parametric TOOLKIT supports the ability to stop and restart a synchronous application within a single session of Creo Parametric. This is particularly useful during development of an application because it enables you to make changes to your source code and retest it without having to restart Creo Parametric and reload your test models. Use the **Auxiliary Applications** dialog box to stop and restart applications.

To make this option available, the registry file (default name `protk.dat` should contain one of the following lines:

```
Multiprocess mode:
     startup spawn
DLL mode:
     startup DLL
```

If you want to be able to stop and restart your Creo Parametric TOOLKIT application within Creo Parametric, you must also add the following statement to the definition of the application in the registry file:

```
     allow_stop TRUE
```

To access the **Auxiliary Applications** dialog box, on the **Home** tab, in the **Utilities** group, click **Auxiliary Applications** or click **Tools ► Auxiliary Applications**. The dialog box displays a list of Creo Parametric TOOLKIT applications identified by the name defined in the `name` statement in its registry file. Only applications that a user can start or stop are displayed. This dialog box also shows the current state of an application and allows an application to be started or stopped.

When a user starts an application from the **Auxiliary Applications** dialog box, Creo Parametric freezes the user interface until the application connects to it.

If you use the `allow_stop` option, you might also set Creo Parametric to not start the Creo Parametric TOOLKIT application until you explicitly request it. To do this, you must add the following statement in your registry file:

```
     delay_start TRUE
```

To start your application in Creo Parametric, choose **Auxiliary Applications** from the **Tools** tab, select your application from the list, then click the **Start** button.

In addition to **Start**, **Stop**, and **Close**, the dialog box includes the following buttons:

- **Register**—Enables you to register a Creo Parametric TOOLKIT application whose registry file was not present when Creo Parametric was started.
- **Info**—Reports the following information about each currently registered Creo Parametric TOOLKIT application:

- The names of the executable file and text directory
- The version number used to build the application
- Whether the application is currently running

There are a few other, less commonly used options in the registry file. All the options are described fully in the Creo Parametric TOOLKIT Registry File on page 2099 appendix.

Note

You can delete registration information on any application that is not running.

When stopping an application, make sure that no application-created menu buttons are current. To do this, before you exit an application you must choose a command that interrupts the current menu command.

Using a Source-Code Debugger on a Creo Parametric TOOLKIT Application

For a full description of how to debug Creo Parametric TOOLKIT applications, see appendix Debugging Creo Parametric TOOLKIT Applications on page 2164. This section also describes optional methods of debugging a Creo Parametric TOOLKIT application.

Unlocking a Creo Parametric TOOLKIT Application

Before you distribute your application executable to the end user, you must unlock it. This enables the end user (your customer) to run your applications without having Creo Parametric TOOLKIT as an option. In Creo Parametric 6.0.0.0 and later you can digitally sign your application.

To unlock your application, enter the following command:

```
<creo_loadpoint>/<datecode>/Parametric/bin/protk_unlock [-c] [-cd]
<path to executables or DLLs to unlock>
```

Note

- The Creo Parametric TOOLKIT application is unlocked even if you do not specify the `-c` option.
- To unlock and digitally sign your application, specify the `-cd` option. Note that it is mandatory to sign your application if you use the `-cd` option. See the section Digitally Signing the Application on page 46, for more information on digitally signing your application.

More than one Creo Parametric TOOLKIT binary file may be supplied on the command line.

Note

Once you have unlocked the executable, you can distribute your application program to Creo Parametric users in accordance with the license agreement.

Using `protk_unlock` requires a valid Creo Parametric TOOLKIT license be present and unused on your license server. If the Creo Parametric license server is configured to add a Creo Parametric TOOLKIT license as a Startup option, `protk_unlock` will cause the license server to hold only the Creo Parametric TOOLKIT option for 15 minutes. The license will not be available for any other development activity or unlocking during this period.

If you use `-cd` option to unlock your application, a message appears asking you to digitally sign your application before using it in Creo.

If the only available Creo Parametric TOOLKIT license is locked to a Creo Parametric license, the entire Creo Parametric license including the Creo Parametric TOOLKIT option will be held for 15 minutes. PTC recommends you configure your Creo Parametric TOOLKIT license option as startup options to avoid tying up your Creo Parametric licenses.

Note

Only one license will be held for the specified time period, even if multiple applications were successfully unlocked.

Unlocking the application may also require one or more advanced licensing options. The `protk_unlock` application will detect whether any functions using advanced licenses are in use in the application, and if so, will make a check for the availability of the advanced license option. If that option is not present, unlocking will not be permitted. If they are present, the unlock will proceed. Advanced options are not held on the license server for any length of time. For more information refer to the Advanced Licensing Options on page 2113 chapter.

If the required licenses are available, the `protk_unlock` application will unlock the application immediately. An unlocked application does not require any of the Creo Parametric TOOLKIT license options to run. Depending on the functionality invoked by the application, it may still require certain Creo Parametric options to work correctly.

Note

Once an application binary has been unlocked, it should not be modified in any way (which includes statically linking the unlocked binary with other libraries after the unlock). The unlocked binary must not be changed or else Creo Parametric will again consider it "locked".

Digitally Signing the Application

In Creo Parametric 6.0.0.0 and later, you can digitally sign your application. Use the standard Microsoft utility SignTool to digitally sign your application. See the Microsoft documentation for more information on this utility and to create the digital certificate.

In Creo Parametric 7.0.0.0 and later, Creo checks signatures of Creo Parametric TOOLKIT applications at load time.

The following configuration options control whether to always allow the users or administrator to determine whether signed or unsigned Creo Parametric TOOLKIT applications should be allowed to always run, to never run, or to prompt the user before running:

- `open_protk_unsigned_apps`—Controls whether unsigned applications can be loaded in a Creo session. It can have the following values:
  - Always—Always loads unsigned applications.
  - Never—Never load unsigned applications.
  - Prompt—Asks the user whether to load unsigned applications.
- `open_protk_signed_apps`—Controls whether signed applications can be loaded in a Creo session. It can have the following values:
  - Always—Always loads signed applications.
  - Never—Never load signed applications.
  - Prompt—Asks the user whether to load signed applications.

Unlock Messages

The following table lists the messages that can be returned when you unlock a Creo Parametric TOOLKIT application.

Message	Cause
<application name>:Successfully unlocked application.	The application is unlocked successfully.
Usage: `protk_unlock` <one or more Creo Parametric TOOLKIT executables or DLLs>	No arguments supplied.
<application name>:ERROR: No READ access	You do not have READ/WRITE access for the

Message	Cause
<application name>:ERROR: No WRITE access	executable.
<application name>:Executable is not a Creo Parametric TOOLKIT application.	The executable is not linked with the Creo Parametric TOOLKIT libraries, or does not use any functions from those libraries.
<application name>:Executable is already unlocked.	The executable is already unlocked.
Error: Licenses do not contain Creo Parametric TOOLKIT License Code.	A requirement for unlocking a Creo Parametric TOOLKIT application.
ERROR: No Creo Parametric licenses are available for the startup command specified	Could not contact the license server.
<application name>:Unlocking this application requires option TOOLKIT-for-3D_Drawings.	The application uses functions requiring an advanced option; and this option is not available. The license option 222, that is, the TOOLKIT-for-3D_Drawings license is not available.
<application name>:Unlocking this application requires option TOOLKIT-for-Mechanica.	The application uses functions requiring an advanced option; and this option is not available. The license option 223, that is, the TOOLKIT-for-Mechanica license is not available.

Structure of a Creo Parametric TOOLKIT Application

The contents of this section refer to the use of synchronous mode. For information on asynchronous mode applications, see the chapter Core: Asynchronous Mode on page 277.

Essential Creo Parametric TOOLKIT Include Files

The only header file you must always include in every source file of your Creo Parametric TOOLKIT application is `ProToolkit.h`. This file must always be present, and must be the first include file because it defines the value of `wchar_t`, the type for characters in a wide string, referenced from many other include files. `ProToolkit.h` also includes these standard include files:

- `stdio.h`
- `string.h`
- `stddef.h`
- `stdlib.h`

Therefore, you do not need to include these header files explicitly in your application.

When you use functions for a particular Creo Parametric TOOLKIT object, you should always include the header file that contains the function prototypes for those functions. If you do not do this, or omit some, you lose the benefit of function argument type-checking during compilation. The header file

ProObjects.h, which contains the declarations of the object handles, is included indirectly in each of the header files that contains function prototypes, and so does not need to be included explicitly.

For example, if you are using the function ProSurfaceAreaEval(), you should include the file ProSurface.h, which contains the prototype of that function, but you do not need to include ProObjects.h in order to see the definition of ProSurface, because ProObjects.h is included in ProSurface.h.

Core of a Creo Parametric TOOLKIT Application

Functions Introduced:

- **user_initialize()**
- **ProEngineerDisplaydatecodeGet()**
- **user_terminate()**

A Creo Parametric TOOLKIT application must always contain the functions user_initialize() and user_terminate(). These functions have the prefix "user_" because they are written by the Creo Parametric TOOLKIT application developer, but they are called from within Creo Parametric at the start and end of the session.

The function user_initialize() initializes a synchronous-mode Creo Parametric TOOLKIT application. This function must be present in any synchronous mode application in order for it to be loaded into Creo Parametric. Use this function to setup user interface additions, or to run the commands required for a non-interactive application. user_initialize() is called after the Creo Parametric application has been initialized and the graphics window has been created. It should contain any initializations that your Creo Parametric TOOLKIT application needs, including any modification of Creo Parametric menus (such as adding new buttons).

Note

- user_initialize() must contain at least one Creo Parametric TOOLKIT API call. Failure to do so causes the Creo Parametric TOOLKIT application to fail and return PRO_TK_GENERAL_ERROR.
- When coding a Creo Parametric TOOLKIT application in C++ you must declare the function user_initialize() as extern "C".

The user_initialize() function is called with a number of optional arguments that can add to your function definition. All input and output arguments to this function are optional and do not need to be in the function signature. These

arguments provide information about command-line arguments entered when Creo Parametric was invoked, and the revision and build number of the Creo Parametric in session. Refer to section the section user_initialize() Arguments on page 51 for more information on the function arguments.

The initialization function must return 0 to indicate that the Creo Parametric TOOLKIT application was initialized successfully. Any other return value will be interpreted as a failure, and the system will notify the Creo Parametric user that the Creo Parametric TOOLKIT application failed. Use the optional output argument to `user_initialize()` to specify the wording of this error message.

The call to Creo Parametric TOOLKIT application using the function `user_initialize()` is delayed until Creo Platform Agent is loaded. When you install a Creo application, an appropriate version of the Creo Platform Agent also gets installed. The Platform Agent is used for all browser-related functionalities, which includes interaction with Windchill. So, to run a Creo Parametric TOOLKIT application which interacts with Windchill using functions, Creo Platform Agent must be initialized first before running the Creo Parametric TOOLKIT application.

The Creo Platform Agent is always initialized first by default. The Creo Parametric TOOLKIT applications are delayed until the Platform Agent is fully initialized. The Platform Agent must load in a maximum of 60 seconds, beyond which the agent will time out. If the Platform Agent fails to load, the Creo Parametric TOOLKIT application will not start and an error message is displayed. You can override this behavior and start the Creo Parametric TOOLKIT application, even if the Platform Agent has not loaded using the following environment variables. The valid values for these variables are true and false.

- `PROTK_DELAYINIT_NO_DELAY`—Initiates Creo Platform Agent. However, Creo Parametric TOOLKIT applications are initiated, without waiting for Platform Agent to load.
- `PROTK_DELAYINIT_ALWAYS_INIT`—Waits for Creo Platform Agent to load. However, it initiates the Creo Parametric TOOLKIT application even if Creo Platform Agent fails to load or times out.

Note

If both the variables are set, then the environment variable `PROTK_DELAYINIT_NO_DELAY` takes precedence.

Note

The Creo Parametric visible datecode format has changed. `user_initialize()` continues to receive the classic format based on the year and week of the Creo Parametric build.

The function `ProEngineerDisplaydatecodeGet()` returns the user-visible datecode string from Creo Parametric. Applications that present a datecode string to users in messages and information should use the new format for the Creo Parametric displayed datecode.

The function `user_terminate()` is called at the end of the Creo Parametric session, after the user selects **Yes** on the **Exit** confirmation dialog box. Its return type is void.

Note

When coding a Creo Parametric TOOLKIT application in C++ you must declare the function `user_terminate()` as `extern "C"`.

The following example is the empty core of a Creo Parametric TOOLKIT application. This code should always be the starting point of each new application you develop.

```
#include "ProToolkit.h"
int user_initialize()
{
    return (0);
}
void user_terminate()
{
}
```

If you use the options to start and stop a multiprocess-mode Creo Parametric TOOLKIT application within a Creo Parametric session, `user_initialize()` and `user_terminate()` are called upon starting and stopping the Creo Parametric TOOLKIT process only. However, any menu modifications defined in `user_initialize()` will be made, even if this involves repainting menus that are already displayed. All of these modifications will be reset when the Creo Parametric TOOLKIT application is stopped.

user_initialize() Arguments

`user_initialize()` is called with a number of input and output arguments. As always in C, if you don't need to use an argument, your function does not need to declare it, provided that it declares all the arguments up to the last one used.

The input arguments are:

int arg_num	**Number of command-line arguments.**
`char *argc[]`	Command-line arguments passed by Creo Parametric. (See further explanation below.)
`char* version`	Release name of the Creo Parametric being used. Note: From Pro/ENGINEER Wildfire 4.0 onwards applications built with libraries older than Pro/ENGINEER 2001 will not run. You must recompile these applications with later versions of the Pro/TOOLKIT libraries.
`char* build`	The build number of the Creo Parametric being used.

The output argument is:

`wchar_t err_buff[80]`	The text of an error message passed to Creo Parametric if the Creo Parametric TOOLKIT fails to initialize. Creo Parametric displays this text when it reports the Creo Parametric TOOLKIT failure (if `user_initialize()` returns non-zero).

The first command-line argument passed to Creo Parametric TOOLKIT is the same one seen by Creo Parametric; that is, it is the name of the Creo Parametric executable. The remaining command-line arguments passed to `user_initialize()` are a subset of those given on the command line that invoked Creo Parametric. The rule is that Creo Parametric passes on to `user_initialize()` any command-line argument that starts with a "+", or with a "-" followed by an upper-case character.

For example, these command-line arguments will be passed to Creo Parametric TOOLKIT:

```
+batch=mybatchfile.txt
-Level=expert
```

Command-line arguments such as `-g:no_graphics` are interpreted by Creo Parametric but not passed on to Creo Parametric TOOLKIT.

Threading in Creo Parametric TOOLKIT Applications

Calling Creo Parametric TOOLKIT applications from within multiple threads of any application in any mode is not supported. Extra threads created by the application are to be used only for completion of tasks that do not directly call the Creo Parametric TOOLKIT functions.

Function Introduced:

- **ProEngineerMultithreadModeEnable()**

From Creo Parametric 3.0 onward, the function `ProEngineerMultithreadModeEnable()` has been deprecated. Multithreading is now always supported in Creo Parametric TOOLKIT applications, without the need to call the multithreading function, when the application creates additional threads for processing.

Call the function `ProEngineerMultithreadModeEnable()` from within the initialization function `user_initialize()`, if your Creo Parametric TOOLKIT application creates additional threads for processing. This function notifies Creo Parametric to execute in the multithread enabled mode. Running in this mode eliminates the possibility of a memory corruption due to interaction between Creo Parametric's thread libraries and the threads created by your application. This function does not work for multiprocess and asynchronous mode applications.

Note

Running Creo Parametric in the multithread enabled mode may slow down performance. Therefore, `ProEngineerMultithreadModeEnable()` should be used only for applications that actually create multiple threads.

Using Creo Parametric TOOLKIT to Make a Batch Creo Parametric Session

Function Introduced:

- **ProEngineerEnd()**

If you want to use your Creo Parametric TOOLKIT application to perform operations on Creo Parametric objects that do not require interaction with the user, you can make all the necessary calls to Creo Parametric TOOLKIT functions in `user_initialize()`. When your operations are complete, call the function `ProEngineerEnd()` to terminate the Creo Parametric session.

A useful technique when designing a batch-mode Creo Parametric TOOLKIT application is to use command-line arguments to Creo Parametric as a way of signaling the batch mode and passing in the name of a batch control file. Consider the following command to start Creo Parametric:

```
pro +batch=<filename>
```

In this example, the option will be ignored by Creo Parametric, but will be passed as an input argument to `user_initialize()`. Inside that function, your code can recognize the switch, and get the name of the file that could contain, for example, the names of Creo Parametric models to be processed, and operations to be performed on each one.

A batch-mode operation should also run without displaying any graphics. To ensure that the Creo Parametric main Graphics Window and Message Window are not displayed, you should use either the command-line option `-g:no_graphics` (or the configuration file option "`graphics NO_GRAPHICS`") to turn off the Creo Parametric graphics. See the Creo Parametric Help for more details of these options.

Example 1: Batch Mode Operation

This example shows how to use the arguments to `user_initialize()` and the function `ProEngineerEnd()` to set up a batch mode session of Creo Parametric. The application retrieves a part specified in the Creo Parametric command line, performs an action on it (using the dummy function `UserAddHoles()`), saves the parts, and terminates Creo Parametric.

```
/*====================================================================*\
FUNCTION: UserAddHoles
PURPOSE:  Find the circular datum curves and replace them with
          holes.
\*====================================================================*/
UserAddHoles (ProMdl p_part)
{
/*  .
    .
    .
*/
}
/*====================================================================*\
    Load the part specified by the command line argument, and
    replace its datum curves with holes.
\*====================================================================*/
int user_initialize (int argc, char *argv[])
{
    ProMdl      p_part;
    ProName     name_wchar;
    ProError    err;
    char       *part_name;
/*--------------------------------------------------------------------*\
    Set up the part name from the argument list. Note that the
    Creo Parametric arguments for Creo Parametric TOOLKIT have a leading
    "+" or "-."
\*--------------------------------------------------------------------*/
    part_name = argv[1];
    part_name++;
    ProStringToWstring (name_wchar, part_name);
/*--------------------------------------------------------------------*\
    Retrieve the part.
\*--------------------------------------------------------------------*/
    err = ProMdlRetrieve (name_wchar, PRO_PART, &p_part);
    if (err != PRO_TK_NO_ERROR)
    {
```

```
        printf ("*** Failed to retrieve part %s\n", part_name);
        ProEngineerEnd();
    }
/*--------------------------------------------------------------------*\
    Add the holes to the part.
\*--------------------------------------------------------------------*/
    UserAddHoles (p_part);
/*--------------------------------------------------------------------*\
    Save the part.
\*--------------------------------------------------------------------*/
    ProMdlSave (p_part);
/*--------------------------------------------------------------------*\
    Terminate the Creo Parametric session.
\*--------------------------------------------------------------------*/
    ProEngineerEnd();
    return (0);
}
/*====================================================================*\
FUNCTION: user_terminate()
PURPOSE:  Report successful termination of the program.
\*====================================================================*/
void user_terminate()
{
    printf ("Creo Parametric TOOLKIT application terminated successfully\n");
}
```

Creo Parametric TOOLKIT Support for Creo Applications

Creo Parametric TOOLKIT applications in synchronous and asynchronous modes are supported only with the Creo Parametric application. They are not supported with the other Creo applications, such as, Creo Layout, Creo Simulate, and so on.

In the asynchronous mode, the functions `ProEngineerConnect()` and `ProEngineerStart()` return an error when the Creo Parametric TOOLKIT application attempts to connect to a Creo application other than Creo Parametric.

For Creo Parametric TOOLKIT applications in synchronous mode, the non-Creo Parametric applications ignore the Toolkit registry files without any warnings. The **Auxiliary Applications** dialog box is also not available within the non-Creo Parametric applications.

User-Supplied Main

Function Introduced:

- **ProToolkitMain()**

In synchronous mode, the `main()` function of the Creo Parametric TOOLKIT program is not written by you, the application developer. In DLL mode, the `main()` is the root of the Creo Parametric program itself; in multiprocess synchronous mode, the `main()` is taken from the Creo Parametric TOOLKIT library, and its job is to set up the communication channel with the separate Creo Parametric executable.

If you are using a language such as C++ in your Creo Parametric TOOLKIT application, it can be advantageous to compile the `main()` function with the C++ compiler to ensure that the program structure is correct for C++. In DLL mode, you cannot do this because you do not have access to the Creo Parametric `main()`. But in multiprocess mode, you can substitute the Creo Parametric TOOLKIT `main()` with your own, if you observe the following rules:

- Your `main()` must call the function `ProToolkitMain()` as its last statement. This function contains all the necessary setup code that needs to be run when the Creo Parametric TOOLKIT application starts up in multiprocess mode.
- You must pass on the *argc* and *argv* arguments input to `main()` as the input arguments to `ProToolkitMain()` without modifying them in any way.
- You cannot make calls to any other Creo Parametric TOOLKIT functions before the call to `ProToolkitMain()`, because the communications with Creo Parametric have not yet been set up. You may, however, make other non-Creo Parametric TOOLKIT function calls before calling `ProToolkitMain()`.

The following example shows a user-defined `main()` for use in multiprocess mode.

```
#include "ProToolkit.h"
main(
    int argc,
    char *argv[])
{
    .
    .
    .
    ProToolkitMain (argc, argv);
    /* The program exits from within ProToolkitMain().
       Any code here is not executed. */
}
```

Asynchronous Mode

For more information on the asynchronous mode, see the chapter Core: Asynchronous Mode on page 277.

Creo Parametric TOOLKIT Techniques

This section describes the basic techniques you use when writing Creo Parametric TOOLKIT applications. The topics are as follows:

Also see the Visit Functions on page 62 section for information on techniques used when writing Creo Parametric TOOLKIT applications.

Object Handles

Each object in Creo Parametric TOOLKIT has a corresponding C typedef, called a "handle", whose name is always the name of the object itself with the prefix "Pro." The handle is used as the type for all variables and arguments that refer to an object of that type. For example, any Creo Parametric TOOLKIT function that performs an action on a solid has an input argument of type `ProSolid`.

Handles are classified into two types, depending on the way in which they are defined and have to be used. The two types are opaque handle (OHandle) and database handle (DHandle). The following sections describe these handles in detail.

OHandles

The simplest way to reference an object in Creo Parametric is to use the memory address of the Creo Parametric data structure that describes that object. To prevent the Creo Parametric TOOLKIT application from accessing the content of the data structure for the object directly, the declaration of the structure is not provided. For example, the object handle `ProSurface` is defined as follows:

```
typedef struct geom* ProSurface;
```

The structure `struct geom` is used to describe a surface in Creo Parametric, but the declaration of the structure is not included in Creo Parametric TOOLKIT. This type of handle is called an opaque handle or opaque pointer for this reason.

Opaque handles have the advantage of simplicity and efficiency—they can be directly dereferenced inside the Creo Parametric TOOLKIT function without any searching. They can also reference items that are transient and not in the Creo Parametric database at all, such as the surfaces and edges that result from an interference volume calculation.

Other examples of Creo Parametric TOOLKIT objects that are given OHandles are as follows:

```
typedef void* ProMdl;
typedef struct curve_header* ProEdge;
typedef struct sld_part* ProSolid;
typedef struct entity* ProPoint;
```

```
typedef struct entity* ProAxis;
typedef struct entity* ProCsys;
typedef struct entity* ProCurve;
```

Because opaque handles are just memory pointers, they suffer the disadvantage of all pointers in that they are volatile—they become invalid if the database object they refer to moves to a different memory location. For example, a `ProSurface` handle (a pointer to a Creo Parametric surface) may become invalid after regeneration of the owning part (because its memory has been reallocated).

However, most of the Creo Parametric structures referenced by opaque handles contain an integer identifier that is unique for items of that type within the owning model. This identifier retains its value through the whole life of that item, even between sessions of Creo Parametric. Creo Parametric TOOLKIT provides functions such as `ProSurfaceIdGet()` and `ProAxisIdGet()` that enable your application to use these identifiers as a persistent way to reference objects. These integer identifiers are also used in DHandles, described in the following section.

In the case of models, it is the name and type that are persistent. The functions `ProMdlMdlnameGet()` and `ProMdlTypeGet()` provide the name and type of a model, given its opaque handle.

DHandles

A further limitation of opaque handles is that they can be too specific in cases where the action you want to perform is more generic. For example, a function that provides the name of a geometrical item should, ideally, be able to act on any of the geometry objects (`ProSurface`, `ProEdge`, `ProCsys`, and so on). However, the opaque handles for those different geometry items are not mutually compatible, so the Creo Parametric TOOLKIT function would also need to know the type of the object before it could internally de-reference the opaque pointer.

To solve this problem, Creo Parametric TOOLKIT defines a new, generic object type in these cases and declares it using a data handle, or DHandle. A DHandle is an explicit data structure that carries just enough information to identify a database item uniquely: the type, integer identifier, and handle to the owning model. Because the DHandle must contain the integer identifier (not the too-specific opaque handle), it also has the advantage of being persistent.

The most important examples of DHandles are `ProGeomitem`, which is the generic type for the geometry items previously mentioned, and `ProModelitem`, which is an even more generic object that includes `ProGeomitem`.

The declaration is as follows:

```
typedef struct pro_model_item
{
   ProType  type;
   int      id;
   ProMdl   owner;
```

```
} ProModelitem, ProGeomitem;
```

Note

Although the field `owner` is defined using the OHandle `ProMdl`, and is therefore strictly speaking volatile, this handle is guaranteed to remain valid while the Creo Parametric model it refers to remains in memory.

The generic object `ProGeomitem` can represent any of the geometrical objects in a solid model, such as `ProSurface`, `ProEdge`, `ProCurve`, and `ProCsys`. The specific object types are said to be "derived from" the most generic type, and also to be "instances" of that type. The object type `ProGeomitem` is in turn an instance of `ProModelitem`, which can represent database items other than geometrical ones.

The generic object types such as `ProModelitem` and `ProGeomitem` are used as inputs to Creo Parametric TOOLKIT functions whose actions are applicable to all of the more specific types of object that are instances of the generic type. For example, the function `ProGeomitemFeatureGet()` has that name because it can act on any type of object that is an instance of `ProGeomitem` `ProSurface`, `ProEdge`, `ProCsys`, and so on. The function `ProModelitemNameGet()` is applicable to a wider range of database objects, not just geometrical ones.

If you have the OHandle to an object, such as `ProSurface`, and you want to call a generic function such as `ProGeomitemFeatureGet()`, you need to convert the OHandle to the more generic DHandle. Functions such as `ProGeomitemInit()` and `ProModelitemInit()` provide this capability. Similarly, you can convert a `ProGeomitem` to a `ProSurface` using the function `ProSurfaceInit()`. These techniques are illustrated in Example 3: Listing Holes in a Model on page 64, in the Visit Functions on page 62 section.

Workspace Handles

When you use Creo Parametric TOOLKIT to create an object in Creo Parametric that contains a lot of information, such as a feature, it is important to be able to set up all of that information before adding the object to the Creo Parametric database. The object-oriented style of Creo Parametric TOOLKIT does not allow explicit access to the contents of such a structure, however. Instead, you must use a special workspace object that is allocated and filled by the Creo Parametric TOOLKIT application using functions provided for that purpose.

The "workspace" is a memory area in Creo Parametric that contains data structures not yet part of the design database.

The workspace object is identified by a handle that contains the address of the memory for the object, which is therefore similar to an OHandle. To distinguish this from handles that refer to objects in the Creo Parametric database, such handles are called workspace handles (WHandles).

Expandable Arrays

Functions Introduced:

- **ProArrayAlloc()**
- **ProArrayFree()**
- **ProArraySizeGet()**
- **ProArraySizeSet()**
- **ProArrayMaxCountGet()**
- **ProArrayObjectAdd()**
- **ProArrayObjectRemove()**

The functions in this section enable you to access a set of programming utilities in general use within Creo Parametric. The utilities fill a need that is common in C and Pascal programming—to provide a storage method that provides the advantages of an array, but without its limitations.

When you use an array for storage for a group of items, you have the advantage over a linked list in that the members are contiguous in memory. This enables you to access a given member using its index in the array. However, if you need to make frequent additions to the members in a way that cannot be predicted (a common situation in MCAE applications), you must reallocate the memory for the array each time.

A common compromise is to allocate the memory in blocks large enough to contain several array members, then reallocate the memory only when a block becomes full. You would choose the size of the blocks such that the frequency of reallocation is significantly reduced, while the amount of unused memory in the last block is acceptably small. The difficulty of this solution is that you would normally need a new set of utilities for each item you want to store as an array, and additional static data for each array to keep track of the number of blocks and the number of members.

The “expandable array” utilities provide a set of functions that can be applied to items of any size. The utilities do this by keeping a private header at the start of the array memory to which the “bookkeeping” information (the number and size of its members, and of the blocks) is written. The pointer your application sees is the address of the first block, not the address of the preceding header.

The importance of the expandable array utilities in a Creo Parametric TOOLKIT application is not only that you can use them for your own arrays, but that you must use them for arrays of data passed between your application and the internals of Creo Parametric through the Creo Parametric TOOLKIT functions.

Note that because the array pointer is not the start of the contiguous memory claimed by the array utility, this pointer is not recognized by the operating system as a valid location for dynamic memory. Therefore, you will cause a fatal error if you try to use the memory management library functions, such as `realloc()` and `free()`.

The basic type used for referring to expandable arrays is `ProArray`, declared as a `void*`.

The function `ProArrayAlloc()` sets up a new expandable array. Its inputs are as follows:

- The initial number of members in the array
- The size, in bytes, of each array member
- Number of objects added to `ProArray` at each memory reallocation. A higher number means more memory is preallocated and fewer reallocations of the `ProArray` are required.

The function outputs a pointer to the contiguous memory that will contain the array members. You can write to that memory to fill the array using the usual memory functions (such as `memcpy()` and `memset()`). If you increase the array size beyond the limit returned by `ProArrayMaxCountGet()`, this function returns an out-of-memory message.

The maximum memory allocated is 2 MB, except for 64–bit platforms where the maximum is twice that.

The function `ProArrayFree()` releases the memory for the specified `ProArray`.

The function `ProArraySizeGet()` tells you how many members are currently in the specified array.

The `ProArraySizeSet()` function enables you to change the number of members in the expandable array. This function is equivalent to `realloc()`.

Function `ProArrayMaxCountGet()`, when given the specified structure size in bytes, returns the maximum number of structure elements a `ProArray` can support for that structure size.

The function `ProArrayObjectAdd()` adds a contiguous set of new members to an array, though not necessarily to the end of the array. The function also sets the contents of the new members. If you increase the array size beyond the limit returned by `ProArrayMaxCountGet()`, this function returns an out-of-memory message.

The function `ProArrayObjectRemove()` removes a member from the array. The member does not necessarily have to be the last member of the array.

Functions `ProArraySizeSet()`, `ProArrayObjectAdd()`, and `ProArrayObjectRemove()` change the size of the array, and might therefore also change its location.

The Creo Parametric TOOLKIT functions use expandable arrays in the following circumstances:

- The function creates a filled, expandable array as its output.
- The function needs a filled, expandable array as its input.
- The function needs an existing expandable array to which to write its output.

An example of the first type of function is the geometry function `ProEdgeVertexdataGet()`, which provides a list of the edges and surfaces that meet at a specified solid vertex. When you have finished using the output, you should free the arrays of edges and surfaces (using the function `ProArrayFree()`).

An example of the second type of function is `ProSolidNoteCreate()`, which creates a design note in a solid. Because the text lines to add to the note are passed in the form of an expandable array, your application must create and fill the array using the functions `ProArrayAlloc()` and `ProArrayObjectAdd()` before you call `ProSolidNoteCreate()`.

An example of the third type of function is `ProElementChildrenGet()`, which gets the number of feature elements that are the children of the specified compound element. The feature elements form a tree that contains all the necessary information about a particular feature. (This function is therefore used in both feature analysis and feature creation.) Before calling `ProElementChildrenGet()`, you must call `ProArrayAlloc()` to create an empty array. You can then use `ProArraySizeGet()` to find out how many elements were added to the array.

There is a fourth case, which is a variation of the first, in which a Creo Parametric TOOLKIT function creates an expandable array as its output the first time it is called in an application, but overwrites the same array on subsequent calls. An example of this is `ProSelect()`, whose output array of `ProSelection` structures must not be freed using `ProArrayFree()`. You must also make sure to copy the contents of the array if you need to use it to make another call to `ProSelect()`.

The conventions are chosen for each function according to its individual needs. For example, `ProElementChildrenGet()` is typically called in a recursive loop to traverse a tree, so the fourth method of allocation would be inconvenient.

The rules for each Creo Parametric TOOLKIT function are documented in the browser.

Example 2: Expandable Arrays

The sample code in `UgFundExpArrays.c` located at `<creo_toolkit_loadpoint>\protk_appls\pt_userguide\ptu_fundament` shows how to use expandable arrays, not as input or output for a Creo Parametric TOOLKIT function, but to create a utility that provides an alternative to a Creo Parametric TOOLKIT visit function. To use Creo Parametric TOOLKIT to access all the features in a solid, you call the function `ProSolidFeatVisit()`. However, you might prefer to use a function that provides an array of handles to all of the features, then traverse this array. This kind of function is called a "collection" function, to distinguish it from a visit function. Although Creo Parametric TOOLKIT does not provide collection functions, you can use the technique demonstrated in the example to write your own.

The utility function `UserFeatureCollect()` passes an empty, expandable array of feature handles as the application data to `ProSolidFeatVisit()`. The visit function `FeatVisitAction()` adds the handle to the visited feature to the array using `ProArrayObjectAdd()`.

Visit Functions

In a Creo Parametric TOOLKIT application, you often want to perform an operation on all the objects that belong to another object, such as all the features in a part, or all the surfaces in a feature. For each case, Creo Parametric TOOLKIT provides an appropriate "visit function." A visit function is an alternative to passing back an array of data.

You write a function that you want to be called for each item (referred to as the "visit action" function) and pass its pointer to the Creo Parametric TOOLKIT visit function. The visit function then calls your visit action function once for each visited item.

Most visit functions also provide for a second callback function, the filter function, which is called for each visited item before the action function. The return value of the filter function controls whether the action function is called. You can use the filter function as a way of visiting only a particular subset of the items in the list.

For example, the visit function for visiting the features in a solid is declared as follows:

```
ProError ProSolidFeatVisit (
  ProSolid                solid,
  ProFeatureVisitAction   visit_action,
  ProFeatureFilterAction  filter_action,
  ProAppData              app_data);
```

The first argument is the handle to the solid (the part or assembly) whose features you want to visit.

The second and third arguments are the visit action function and filter function, respectively.

The type of the final argument, `ProAppData`, is a typedef to a `void*`. This argument is used to pass any type of user-defined application data down to the `visit_action` and `filter_action` functions through the intervening Creo Parametric TOOLKIT layer. You might want to use this as an alternative to allowing global access to the necessary data.

Although you write the visit action and filter functions, they are called from within the Creo Parametric TOOLKIT visit function, so their arguments are defined by Creo Parametric TOOLKIT. To enable the C compiler to check the arguments, Creo Parametric TOOLKIT provides a typedef for each of these functions.

For example, the type for the action function for `ProSolidFeatVisit()` is as follows:

```
typedef ProError (*ProFeatureVisitAction)(
  ProFeature  *feature,
  ProError     status,
  ProAppData   app_data);
```

It takes three arguments:

- The handle to the feature being visited
- The status returned by the preceding call to the filter function
- The application data passed as input to the visit function itself

The type for the filter function is as follows:

```
typedef ProError (*ProFeatureFilterAction)(
  ProFeature   *feature,
  ProAppData    app_data);
```

Its two arguments are the handle to the feature being visited and the application data.

The filter action function should return one of the following values:

- `PRO_TK_CONTINUE`—Do not call the visit action for this object, but continue to visit the subsequent objects.
- Any other value—Call the visit action function for this object and pass the return value as the status input argument.

The visit action function should return one of the following values:

- PRO_TK_NO_ERROR—Continue visiting the other objects in the list.
- PRO_TK_E_NOT_FOUND—For visit functions, this value indicates that no items of the desired type were found and no functions could be visited.
- Any other value (including PRO_TK_CONTINUE)—Terminate the visits. Typically this status is returned from the visit function upon termination, so that the calling function knows the reason that visiting terminated abnormally.

Example 3: Listing Holes in a Model

The sample code in UgFundVisit.c located at <creo_toolkit_loadpoint>\protk_appls\pt_userguide\ptu_fundament demonstrates several of the principles used in Creo Parametric TOOLKIT, including visit functions, the use of Ohandles and Dhandles, and the ProSelection object.

The example shows the function UserDemoHoleList(), which visits the axes in the current part that belong to features of type HOLE. It then writes the axis names and feature identifiers to a file, and highlights the hole features.

The top function, UserDemoHoleList(), calls ProSolidAxisVisit(). The function uses the ProAppData argument to pass to the visit action function, UserDemoAxisAct(), a structure that contains the file pointer and handle to the owning solid.

Support for Creo Model Names and Files Paths

Creo Parametric supports a maximum length of 31 characters for file names of native Creo models. This excludes the file extension. The local file paths can contain a maximum of 260 characters. File paths support some multi-byte characters.

From Creo Parametric 4.0 F000 onward, file names support multi-byte characters.

The file names and file paths support the following multi-byte characters:

- All characters from Unicode number 0800 onward.
- The following characters from Unicode numbers 0000 to 0070F are supported. All the other Unicode characters between 0000 to 0070F are not supported.
  - A to Z
  - a to z
  - 0 to 9
  - _ Underscore
  - – Hyphen

All the Creo Parametric TOOLKIT functions support multi-byte characters in file names and file paths of the models.

Wide Strings

Creo Parametric TOOLKIT, like Creo Parametric, has to work in environments where character strings use codes other than ASCII, and might use a bigger character set than can be coded into the usual 1-byte char type, for example, the Japanese `KANJI` character set.

For this reason, Creo Parametric TOOLKIT uses the type `wchar_t` instead of `char` for all characters and strings that may be visible to the Creo Parametric user. This includes all text messages, keyboard input, file names, and names of all dimensions, parameters, and so on, used within a Creo Parametric object.

Defining wchar_t

Although most platforms supported by Creo Parametric TOOLKIT provide a definition of `wchar_t` in a system include file, not all do. Those that do use definitions of different lengths; some provide definitions that are not suitable for all the character codes supported by Creo Parametric. Therefore, Creo Parametric takes considerable care to make sure it uses a suitable definition of `wchar_t` on each supported platform.

It is essential to make sure your Creo Parametric TOOLKIT application is using the same definition of `wchar_t` as Creo Parametric on each platform your application supports. To make this easier, Creo Parametric TOOLKIT supplies the include file`pro_wchar_t.h`. This file ensures that, if a definition of `wchar_t.h` has not already been made in an earlier include file, one is provided that is consistent with the Creo Parametric definition of the type. Because this file is included by the file `ProToolkit.h`, you should include `ProToolkit.h` as the very first include file in each source file.

Setting the Hardware Type

To make the handling of the wide character type `wchar_t` across different platforms simpler and more reliable, the include file `pro_wchar_t.h` is hardware dependent. It knows which platform is being used from the setting of the environment variable `PRO_MACHINE`; the recognized values are listed in the include file `pro_hardware.h`, included by `pro_wchar_t.h`.

You must make sure that the environment variable `PRO_MACHINE` is set to indicate the type of hardware you are using. Set it to same value used for the makefile macro `PRO_MACHINE` in the makefile taken from the Creo Parametric TOOLKIT loadpoint.

Checking Your Declaration of wchar_t

Function Introduced:

- **ProWcharSizeVerify()**

The function `ProWcharSizeVerify()` checks to make sure you have the correct declaration of `wchar_t`. PTC recommends that you always call this function at the beginning of the `user_initialize()` function (or `main()` in asynchronous mode).

You pass as input the size of your `wchar_t` definition, in bytes, and the function outputs the correct size. It returns `PRO_TK_NO_ERROR` if your size is correct, and `PRO_TK_GENERAL_ERROR` otherwise. You can check for correctness as follows:

```
int proe_wchar_size;
int protk_wchar_size = sizeof (wchar_t);

if (ProWcharSizeVerify (protk_wchar_size, &proe_wchar_size) !=
    PRO_TK_NO_ERROR)
{
    ProMessageDisplay (msgfil, "USER wchar_t size is %0d,
         should be %1d", &protk_wchar_size, &proe_wchar_size);
    return (1);
}
```

String and Widestring Functions

Creo Parametric provides many functions taking as inputs a fixed-length character or wide character array as a string. Due to some platform-specific behavior, PTC recommends that you do not use string literals in lieu of fixed length arrays. Always copy the literal strings to the full size array that the functions accepts as the input.

For example the following function will get warnings on certain platforms because the code expects that the arguments can be modified.

```
ProEngineerStart("proe_path","text_path");
```

where `ProCharPath proe_path=”proe_path”;`

`ProCharPath text_path=”text_path”;`

To overcome this error, it is recommended that you replace the literal strings in the function with defined arrays as follows:

```
ProEngineerStart(proe_path,text_path);
```

Functions Introduced:

- **ProStringToWstring()**
- **ProWstringToString()**

- **ProWstringLengthGet()**
- **ProWstringCopy()**
- **ProWstringCompare()**
- **ProWstringConcatenate()**

Wide character strings are not as easy to manipulate in C as ordinary character strings. In general, there are no functions for wide strings that correspond to the standard C str*() functions. printf() does not have a format for wide strings, and you cannot set a wide string to a literal value in a simple assignment. Because of this, it is frequently convenient to convert wide strings to character strings, and vice versa. This is the purpose of the functions `ProStringToWstring()` and `ProWstringToString()`.

The function `ProWstringLengthGet()` is used to find the length of a widestring.

The function `ProWstringCopy()` copies a widestring into another buffer. You should allocate enough memory in the target setting to perform the copy operation. The number of characters to be copied is provided as input through num_chars. Use `PRO_VALUE_UNUSED` to copy the entire string.

The function `ProWstringCompare()` compares two widestrings for equality. The two widestrings to be compared are given as inputs. The argument num_chars allows comparison of portions of the string, pass `PRO_VALUE_UNUSED` to compare the entire strings.

The function `ProWstringConcatenate()` concatenates two widestrings. You should allocate enough memory in the target string for the copy operation. The number of characters to concatenate is given as input through num_chars. Use `PRO_VALUE_UNSED` to add the entire source string to the target string.

The source code for other useful utilities is located in the file `<TK_LOADPOINT>protk_appls\pt_examples\pt_utils\UtilString.c`

Example 4: String Conversion

The sample code in `UgFundStringConv.c` located at `<creo_toolkit_loadpoint>\protk_appls\pt_userguide\ptu_fundament` uses the function `UsrModelFilenameGet()` to convert wide strings to character strings.

Support for IPv6

Creo 6.0.0.0 and later releases have complete support for Internet Protocol version 6 (IPv6). By default Creo uses IPv6 for IP addressing over the network.

Set the environment variable `PTC_IPV6_MODE` to `yes` for Creo to use the IPv6 protocol for addressing. To use IPv4 communication protocol, set the environment variable to `no`.

If you want to run applications created in Creo Parametric 5.0.0.0 and previous releases in Creo Parametric 6.0.0.0 with IPv6 enabled, you must rebuild the applications.

If the environment variable is set to `no`, that is for IPv4 protocol, these applications continue to work in Creo Parametric 6.0.0.0 without rebuilding.

Accessing LearningConnector

Function Introduced:

- **ProLearningconnectorNotify()**

LearningConnector (LC) is a feature in Creo that allows users to access the Creo training directly from a Creo application. An event triggered in the Creo user interface notifies the LearningConnector, which displays the relevant training topics for the current activity.

The function `ProLearningconnectorNotify()` notifies the LearningConnector that an event has been triggered from a Creo Parametric TOOLKIT application. The function also allows users to notify custom-made trainings other than the standard training provided by PTC for the specified module. The LearningConnector displays the relevant training topics for the specified input arguments. You must read the LearningConnector documentation before using this function. The input arguments are:

- *module*—This is a mandatory argument. Specifies the name of the Creo module that triggers the event.
- *module_info*—This is an optional argument. Specifies additional information about the Creo module that triggers the event. This additional information is used by the event-handling function of LearningConnector.

You can use the function on customized Creo Parametric TOOLKIT widgets also.

2

Core: Models and Model Items

This chapter describes Creo Parametric TOOLKIT modes, models, and model items.

Modes

Functions Introduced:

- **ProModeCurrentGet()**
- **ProSectionIsActive()**

The term “mode” in Creo Parametric refers to the type of model currently being edited by the user. The possible modes are given by the options listed under the command **File ▶ New**.

The `ProMode` object in Creo Parametric TOOLKIT is an enumerated type, declared in the file `ProMode.h`, as are the prototypes for the mode functions.

Find the name of the mode using the function `ProModeCurrentGet()`. The function `ProModeCurrentGet()` outputs the mode in which Creo Parametric is being used, in the form of the `ProMode` enumerated type. If there is no current model—for example, because no model has been retrieved, or because the user has selected **File ▶ Close**—the function returns an error status (`PRO_TK_BAD_CONTEXT`).

The function `ProSectionIsActive()` checks if the sketcher is currently active even if the current mode is part or assembly.

Models

This section describes Creo Parametric TOOLKIT models. The topics are as follows:

The ProMdl Object

A model is a top-level object in a Creo Parametric mode. For example, in Part mode, the model is a part; in Assembly mode, the model is an assembly.

The object `ProMdl` is therefore used for all those functions whose action applies to models of any type, such as file management operations and version stamps.

The declaration of `ProMdl` is as follows:

```
typedef void* ProMdl;
```

Instances of the `ProMdl` object are objects for the more specific Creo Parametric modes. For example, `ProSolid` is an instance of `ProMdl`, and `ProAssembly` and `ProPart` are instances of `ProSolid`. All these object types are represented in Creo Parametric TOOLKIT by opaque handles, and you can make conversions between the types by casting.

Creating Models

Functions Introduced

- **ProSolidMdlnameCreate()**
- **ProMfgMdlCreate()**
- **ProSection2DAlloc()**
- **ProDrawingFromTmpltCreate()**
- **ProDrawingFromTemplateCreate()**
- **ProMdlStartAction()**

Creo Parametric TOOLKIT supports creation of models for Solids, Manufacturing, Section (two-dimensional only), and Drawing.

See Creating a Solid on page 93 for a complete description of `ProSolidMdlnameCreate()`.

For more information on `ProMfgMdlCreate()` see Creating a Manufacturing Model on page 1440.

Allocating a Two-Dimensional Section on page 989 gives more details on `ProSection2DAlloc()`.

Creating Drawings from Templates on page 1227 has more information on the function `ProDrawingFromTemplateCreate()`.

Note

The function `ProDrawingFromTmpltCreate()` will be deprecated in a future release. Use the function `ProDrawingFromTemplateCreate()` instead.

The notification function `ProMdlStartAction()` is a type for a callback function for `PRO_MDL_START`. This function changes the way users can create models by replacing the Creo Parametric model template dialog box with a user-specified action.

The user-specified action contains user-programmed activities that allow customization of new models by applying templates with more inputs than model creation “on-the-fly” or the standard Creo Parametric template.

The callback function is activated after the user selects **OK** from the **File ▸ New** dialog box, but only if the **Use Default Template** checkbox is not selected. The user's application must create a new model of the same type and subtype specified by the callback function.

Setting the configuration option `force_new_file_options_dialog` to `yes` forces the **Use Default Template** button to be hidden, and calls the callback for all models created through the **File ▸ New** dialog.

This function supports all model types.

See Event-driven Programming: Notifications on page 2010 for more data on using callback functions.

Identifying Models

Functions Introduced:

- **ProMdlMdlnameGet()**
- **ProMdlOriginGet()**
- **ProMdlExtensionGet()**
- **ProMdlDirectoryPathGet()**
- **ProMdlTypeGet()**
- **ProMdlDisplaynameGet()**
- **ProMdlCommonnameGet()**
- **ProMdlCommonnameSet()**
- **ProMdlObjectdefaultnameGet()**
- **ProMdlnameInit()**
- **ProMdlIdGet()**
- **ProMdlActiveGet()**
- **ProMdlSubtypeGet()**
- **ProMdlFiletypeGet()**
- **ProFileSubtypeGet()**
- **ProMdlToModelitem()**

The object `ProMdl` is an opaque handle, and is therefore volatile. It cannot be used to refer to models that are not in memory in Creo Parametric, for example. To reference a model in a way that is valid for models not in memory, and also persistent across sessions of Creo Parametric, use the model name and type.

The functions `ProMdlMdlnameGet()` and `ProMdlTypeGet()` provide the name and type of a model, given its `ProMdl` handle. The type of a model is expressed in terms of the enumerated type `ProMdlType`. From Creo Parametric

3.0 onward, this enumerated data type contains an additional value PRO_MDL_CE_SOLID that represents a Layout model. Creo Parametric TOOLKIT functions will only be able to read models of type Layout, but will not be able to pass Layout models as input to other functions. PTC recommends that you review all Creo Parametric TOOLKIT applications that use the enumerated type ProMdlType and modify the code as appropriate to ensure that the applications work correctly.

The function ProMdlOriginGet() retrieves the full source path of the model, that is, the path from where the specified model has been opened. It returns NULL if the specified model is new in the session, and has not been saved. For instances, it returns the full path of the generic model.

The function ProMdlExtensionGet() retrieves the file extension for the specified model.

The function ProMdlDirectoryPathGet() returns the file path where the specified model would be saved. It specifies the target home directory for the model.

The function ProMdlDisplaynameGet() returns the name of the model, which is displayed in the Creo Parametric user interface. The name is displayed in the graphics area, such as, the model tree, window title, and so on. If the model is an instance of native Creo model, the display name is the instance name. For configurations or instances of non-native models, the display name consists of the model name along with the configuration name or instance name. If you specify the input argument *include_ext* as PRO_B_TRUE, then the display name returned by the function also includes the file extension of the model.

The functions ProMdlCommonnameGet() and ProMdlCommonnameSet() obtain and assign the common name of a model, respectively. This name is used to identify the model in a Product Database Management system such as Windchill PDMLink.

Note

ProMdlCommonnameSet() can modify the name only for models that are not yet owned by Windchill PDMLink, or in certain situations if the configuration option let_proe_rename_pdm_objects is set to yes.

The function ProMdlObjectdefaultnameGet() returns the next available default name for a given model type. The type of the model is specified by the enumerated type ProType and has one of the following values:

- PRO_PART
- PRO_ASSEMBLY
- PRO_CABLE

- PRO_DRAWING
- PRO_REPORT
- PRO_2DSECTION
- PRO_3DSECTION
- PRO_LAYOUT
- PRO_DWGFORM
- PRO_MARKUP

Note

For each of the above types, ProMdlObjectdefaultnameGet() returns the next available default name, for example PRT00# for PRO_PART or DRW00# for PRO_DRAWING, where # specifies the part or drawing number. This number depends on the following factors:

- Models present in the active Creo Parametric session
- Files in the current working directory
- Connection to an active server with the autonumber option enabled

Thus, if no object with the specified name is actually created, the next time the same name is returned; otherwise the next available name is returned.

The function ProMdlnameInit() does the opposite, and provides a valid ProMdl handle for a given name and type. The function fails if the specified model is not in memory in the Creo Parametric session.

A third way to identify a model is by an integer identifier. Unlike the integer identifiers of objects within a model, such as surfaces and edges, the identifier of a model is not persistent between Creo Parametric sessions. The function ProMdlIdGet() provides the identifier of a model, given its ProMdl handle.

The function ProMdlActiveGet() retrieves the model handle ProMdl for an active Creo Parametric object.

The function ProMdlSubtypeGet() provides the subtype (such as sheet metal) of a specified model. Valid model subtypes are Part, Assembly, or Manufacturing. This is like finding subtypes at the Creo Parametric **File ► New ► Model Type** menu.

The function ProMdlFiletypeGet() retrieves the file type of the specified model using the enumerated data type ProMdlfileType.

The function `ProFileSubtypeGet()` retrieves the following information when you specify the path to a file as the input argument. The output arguments are:

- *file_type*—Specifies the file type using the enumerated data type `ProMdlfileType`. For native Creo models, the file type and model type are the same.
- *type*—Specifies the model type using the enumerated data type `ProMdlType`.
- *subtype*—Specifies the subtype of the model using the enumerated data type `ProMdlsubtype`. For model types that do not have subtypes, the argument returns `PROMDLSTYPE_NONE`.

The function `ProMdlToModelitem()` is used only when you need to represent the model as a `ProModelitem` object—the first step in building a `ProSelection` object that describes the role of a model in a parent assembly. Model item objects are described later in this chapter. See the chapter Fundamentals on page 22 for information on the `ProSelection` object.

Example 1: Finding the Handle to a Model

The following example shows how to find a model handle, given its name and type.

```
ProName  name;
ProType  type;
ProMdl   part;
ProError status;
ProStringToWstring (name, "PRT0001");
type = PRO_PART;
status = ProMdlnameInit (name, type, &);
```

Surface Properties of Models

Functions Introduced:

- **ProMdlVisibleSideAppearancepropsGet()**
- **ProMdlVisibleSideAppearancepropsSet()**
- **ProMdlVisibleSideTexturepropsGet()**
- **ProMdlVisibleSideTexturepropsSet()**
- **ProMdlLockGet()**
- **ProMdlLockSet()**
- **ProMdlVisibleSideTextureplacementpropsGet()**
- **ProMdlVisibleSideTextureplacementpropsSet()**

From Creo Parametric5.0.0.0 onwards, the following functions have been deprecated:

- `ProMdlVisibleAppearancepropsGet()`
- `ProMdlVisibleAppearancepropsSet()`
- `ProMdlVisibleTexturepropsGet()`
- `ProMdlVisibleTexturepropsSet()`
- `ProMdlVisibleTextureplacementpropsGet()`
- `ProMdlVisibleTextureplacementpropsSet()`

The functions described in this section enable you to retrieve and set the surface and texture properties of models. You can retrieve and set these properties for any level in the model hierarchy. For assemblies, set the `owner` of `ProModelitem` as the top-level assembly. These properties may or may not be visible in the user interface depending on the properties set by the higher level assembly.

Use the function `ProMdlVisibleSideAppearancepropsGet()` to retrieve the surface properties for a specified part, assembly component, subassembly, specified side of a quilt or surface using the `ProSurfaceAppearanceProps` data structure. Refer to the section Surface Properties on page 496, for more information on `ProSurfaceAppearanceProps` data structure. The input arguments are:

- *item*—Specifies a `ProAsmitem` object that represents the part, assembly component, subassembly, quilt, or surface.
- *surface_side*—Specifies the direction of the side for the surface or quilt. Pass the value as 0 to specify the side which is along the surface normal. Pass 1 to specify the side opposite to surface normal.

Use the function `ProMdlVisibleSideAppearancepropsSet()` to set the surface properties for a specified element. To see the changes in the Creo Parametric user interface, call the function `ProWindowRepaint()` after `ProMdlVisibleSideAppearancepropsSet()`. To set the default surface appearance properties, pass the value of the input argument `appearance_properties` as `NULL`.

Use the functions `ProMdlVisibleSideTexturepropsGet()` and `ProMdlVisibleSideTexturepropsSet()` to apply textures to the surface. These functions use the `ProSurfaceTextureProps` data structure to retrieve and set the texture properties of the surface for a specified element. Refer to the section Surface Properties on page 496, for more information on `ProSurfaceTextureProps` data structure.

Use the functions `ProMdlVisibleSideTextureplacementpropsGet()` and `ProMdlVisibleSideTextureplacementpropsSet()` to retrieve and set the properties related to the placing of surface texture for the specified element. These functions use the `ProSurfaceTexturePlacementProps` data structure to define the placement properties.

Refer to the section Surface Properties on page 496, for more information on `ProSurfaceTexturePlacementProps` data structure.

The functions `ProMdlLockGet()` and `ProMdlLockSet()` get and set the lock/unlock state of the model. The function `ProMdlLockGet()` returns `PRO_B_TRUE` if the model is locked and `PRO_B_FALSE` if it is unlocked.

The input arguments to the function `ProMdlLockSet()` follow:

- *model*—The model to be locked or unlocked.
- *lock*—Pass the value as `PRO_B_TRUE` to lock the model and `PRO_B_FALSE` to unlock it.

Models in Session

Functions Introduced:

- **ProSessionMdlList()**
- **ProMdlCurrentGet()**
- **ProMdlDependenciesDataList()**
- **ProMdlDependenciesCleanup()**
- **ProMdlDeclaredDataList()**
- **ProMdlModificationVerify()**
- **ProMdlIsModifiable()**
- **ProMdlIsEmbeddedName()**
- **ProMdlVisibleGet()**

The function `ProSessionMdlList()` provides an array of `ProMdl` handles to models of a specified type currently in memory.

The function `ProMdlCurrentGet()` provides the `ProMdl` handle to the model currently being edited by the user.

The function `ProMdlDependenciesDataList()` provides an array of `ProMdl` handles to the models in memory upon which a specified model depends. One model depends on another if its contents reference that model in some way. For example, an assembly depends on the models that form its components, and a drawing model depends on the solid models contained in it. Sometimes, two models can be mutually dependent, such as when a model feature references a geometry item in a parent assembly. Clean the dependencies in the database using the function `ProMdlDependenciesCleanup()` to get an accurate list of dependencies for an object in the Creo Parametric workspace.

Use the function `ProMdlDependenciesCleanup()` to clean the dependencies for an object in the Creo Parametric workspace.

Note

Do not call the function `ProMdlDependenciesCleanup()` during operations that alter the dependencies, such as, restructuring components and creating or redefining features.

The function `ProMdlDeclaredDataList()` provides an array of `ProMdl` handles to first-level notebook models that have been declared to a specified solid model.

The function `ProMdlModificationVerify()` tells you whether a specified model in memory has been modified since it was last saved or retrieved. See the section Version Stamps on page 83 for a more flexible way of keeping track of changes to a model.

The function `ProMdlIsModifiable()` checks if the specified model is modifiable.

The function `ProMdlIsEmbeddedName()` checks if the specified model name or full path that includes the model name is an embedded model name. The output argument `is_embedded_name` returns `PRO_B_TRUE` if the model name is an embedded name, `PRO_B_FALSE` if not.

In intersected embedded components, the name of the embedded model cannot be used for file operations. If you are using the embedded name while creating a new application, the file operation might fail. In such case, you must use the generic or visible model name.

The function `ProMdlVisibleGet()` returns the handle to the generic or visible model for the specified model. The function returns an error `PRO_TK_E_NOT_FOUND`, when the generic or visible model does not exist or is not found in the Creo Parametric session.

When an embedded component is extracted, a copy of the embedded solid is created and the embedded component model is replaced by the new non-embedded copy. In this extract operation, the original embedded model is erased and the name of the model is changed.

For embed operations, if a model has embedded components, a copy of the existing embedded models is created under the currently embedded model.

File Management Operations

Functions Introduced:

- **ProMdlnameCopy()**
- **ProMdlfileMdlnameCopy()**
- **ProMdlnameRetrieve()**
- **ProMdlMultipleRetrieve()**
- **ProSolidRetrievalErrorsGet()**
- **ProMdlSave()**
- **ProMdlIsSaveAllowed()**
- **ProMdlErase()**
- **ProMdlEraseNotDisplayed()**
- **ProMdlEraseAll()**
- **ProMdlnameRename()**
- **ProMdlnameBackup()**
- **ProMdlDelete()**
- **ProMdlLocationIsStandard()**

These functions perform the same actions as the corresponding Creo Parametric file management commands, with the following exceptions:

- `ProMdlnameCopy()` and `ProMdlfileMdlnameCopy()` are equivalent to the **Save As** command in the **File** pull-down menu of the Creo Parametric menu bar. `ProMdlnameCopy()` takes the model handle as input, whereas `ProMdlfileMdlnameCopy()` takes the type and name of the model to copy.
- `ProMdlnameRetrieve()` retrieves the model into memory, but does not display it or make it the current model.
- `ProMdlMultipleRetrieve()` retrieves multiple models into memory. Use the *ui_flag* parameter to set model display to on or off.
- `ProSolidRetrievalErrorsGet()` returns the data structure containing errors that occur during model retrieval. While retrieving a complex assembly, Creo Parametric sometimes encounters errors in retrieving particular components and assembling them appropriately in the assembly. In the user interface, you are informed of errors as they occur, through a dialog box. In Creo Parametric TOOLKIT, the retrieval functions automatically suppress or freeze problem components and return `PRO_TK_NO_ERROR`. To know whether errors have occurred during retrieval, use the function `ProSolidRetrievalErrorsGet()`. The errors are returned as the elements of the `ProSolidretrievalerrs` array. The retrieval error information must be obtained immediately after a call to the `ProMdlnameRetrieve()` or equivalent retrieval function.

- `ProMdlSave()` saves the specified model to disk. For drawings, sketches and other 2D model types, to save the graphics data, you must display it. Call the function `ProMdlDisplay()` before `ProMdlSave()`, so that the graphics data is saved along with the geometry for the model.
- `ProMdlIsSaveAllowed()` checks whether a given model can be saved.
- `ProMdlEraseNotDisplayed()` erases all the models that are not referenced in a window from the current session.
- `ProMdlErase()` erases the specified model from memory.
- `ProMdlEraseAll()` erases a model and all the models that it uses, except those that have cyclic dependencies (that is, models used by other models in the session). For example, `ProMdlEraseAll()` recursively erases all subassemblies of an assembly and all solids referenced from a drawing. This function also works in cases where some models to be erased have mutual dependencies, but only if the erased models are not used by other models.

  However, while erasing an active model, `ProMdlErase()` and `ProMdlEraseAll()` only clear the graphic display immediately, they do not clear the data in the memory until the control returns to Creo Parametric from the Creo Parametric TOOLKIT application. Therefore, after calling them the control must be returned to Creo Parametric before calling any other function, otherwise the behavior of Creo Parametric may be unpredictable.

The function `ProMdlLocationIsStandard()` checks if the specified model was opened from a standard location. A standard file location can be the local disk or a mapped drive on a remote computer. The Universal Naming Convention (UNC) path for network drives is also considered as a standard path if the value for **DisableUNCCheck** is set to True for the key `HKEY_CURRENT_USER\Software\Microsoft\Command Processor`, in the registry file. The function returns:

- `PRO_B_TRUE` when the file is loaded from a standard file location.
- `PRO_B_FALSE` when the file is loaded from a nonstandard file location, such as, http, ftp, Design Exploration mode, and so on.

Model Items

A “model item” is a generic object used to represent any item contained in any type of model, for the purpose of functions whose actions are applicable to all these types of item. (Some items, such as “version stamp,” retain their own object types.)

The object type `ProModelitem` is a DHandle (data handle), a structure that contains the item type, the persistent integer identifier of the item, and the handle to the owning object.

The object `ProGeomitem`, a generic geometrical object described later in this guide, is an instance of `ProModelitem`, and is a DHandle that shares the same type declaration. Therefore, the functions in this section are also directly applicable to `ProGeomitem` objects.

The typedef for the `ProModelitem` data handle is as follows:

```
typedef struct pro_model_item
{
  ProType   type;
  int       id;
  ProMdl    owner;
} ProModelitem
```

Functions Introduced:

- **ProModelitemByNameInit()**
- **ProModelitemInit()**
- **ProModelitemMdlGet()**
- **ProModelitemDefaultnameGet()**
- **ProModelitemNameGet()**
- **ProModelitemNameSet()**
- **ProModelitemNameCanChange()**
- **ProModelitemUsernameDelete()**
- **ProModelitemHide()**
- **ProModelitemUnhide()**
- **ProModelitemIsHidden()**

The function `ProModelitemByNameInit()` returns a pointer to an item (structure), given the name and the type of the item. The valid item types are:

- Edge
- Surface
- Feature
- Co-ordinate System
- Axis
- Point
- Quilt
- Curve
- Layer
- Note

The function ProModelitemInit() is used to generate a ProModelitem object from the information contained in the structure. You can create such a structure directly, but using this function you can also confirm the existence of the item in the model database.

The function ProModelitemMdlGet() extracts the ProMdl handle from the structure.

The function ProModelitemDefaultnameGet() gets the default name for a new model item of a particular type if it was created taking the model handle as input.

The two functions ProModelitemNameGet() and ProModelitemNameSet() read and set the name of the Creo Parametric database object referred to by the model item. These functions are therefore applicable to all the instances of ProModelitem, such as ProGeomitem and all its instances, including ProSurface, ProEdge, ProCsys, and ProAxis.

Note

In addition to notes of the type PRO_NOTE, the functions ProModelitemNameGet() and ProModelitemNameSet() can be used to read and set the name of the following annotation types:

- Driving or driven dimension of the type PRO_DIMENSION
- Reference dimension of the type PRO_REF_DIMENSION
- Symbol instance of the type PRO_SYMBOL_INSTANCE
- Surface finish of the type PRO_SURF_FIN
- Geometric tolerance of the type PRO_GTOL
- Set datum tag of the type PRO_SET_DATUM_TAG (applicable only for ProModelitemNameGet())

The function ProModelitemNameCanChange() identifies whether the name of the model item can be modified by the user or by Creo Parametric TOOLKIT.

The function ProModelitemUsernameDelete() deletes the user-defined name of the model item from the Creo Parametric database.

The functions ProModelitemHide() and ProModelitemUnhide() are equivalent to the **View ► Hide** and **View ► Unhide** commands in the Creo Parametric menu, respectively. ProModelitemHide() hides the specified model item, whereas ProModelitemUnhide() unhides the model item.

The function ProModelitemIsHidden() identifies if the specified model item is hidden.

Example 2: Renaming a Selected Surface

The sample code in `UgGeomSurfRename.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_geometry` shows how to use the functions `ProModelitemNameGet()` and `ProModelitemNameSet()`. See the Core: 3D Geometry on page 170 chapter for an explanation of `ProSurface` and its functions.

Version Stamps

The version stamp object provides a way of keeping track of changes in a Creo Parametric model to which your Creo Parametric TOOLKIT application may need to respond. Creo Parametric models and features contain an internal version stamp incremented each time some design change is made to that model or feature. The functions in this section enable you to read version stamps in order to look for design changes.

The version stamp object is called `ProWVerstamp` because it is a WHandle, or workspace handle. It is a workspace handle because the data structure it references is not the one in the Creo Parametric database, but a copy taken from it, which is private to the Creo Parametric TOOLKIT application.

Functions Introduced:

- **ProMdlVerstampGet()**
- **ProFeatureVerstampGet()**
- **ProVerstampAlloc()**
- **ProVerstampFree()**
- **ProVerstampStringGet()**
- **ProVerstampStringFree()**
- **ProStringVerstampGet()**
- **ProVerstampEqual()**

The functions `ProMdlVerstampGet()` and `ProFeatureVerstampGet()` enable you to make a workspace copy of the version stamp on a particular model or feature. The function `ProMdlVerstampGet()` is currently applicable to solids only (parts or assemblies). Both of these functions allocate the space for the workspace object internally. After using the contents of the version stamp object, you can free the workspace memory using `ProVerstampFree()`.

If you want to store a copy of a version stamp to compare to a newly read version later, you should use the nonvolatile representation, which is a C string. The function `ProVerstampStringGet()` allocates and fills a string that represents the contents of the specified `ProWVerstamp` object. The

`ProStringVerstampGet()` function performs the reverse translation: it allocates a new `ProWVerstamp` object and fills it by copying the specified C string.

The function `ProVerstampEqual()` compares two `ProWVerstamp` objects to tell you whether the version stamps they represent are equal.

Note

The version stamp on a feature can change not only when the feature definition changes, but also when the feature geometry changes as a result of a change to a parent feature.

Layers

Creo Parametric TOOLKIT implements two data types that enable access to layer information in Creo Parametric:

- `ProLayer`—A DHandle that identifies a layer. The `ProLayer` object is an instance of `ProModelitem`.
- `ProLayerItem`—A DHandle that identifies a layer item. The valid types of layer item are contained in the enumerated type `ProLayerType`.

Functions Introduced:

- **ProMdlLayerGet()**
- **ProMdlLayerVisit()**
- **ProMdlLayersCollect()**
- **ProLayerCreate()**
- **ProLayerDelete()**
- **ProLayerItemsGet()**
- **ProLayerItemsPopulate()**
- **ProLayeritemarrayFree()**
- **ProLayerItemInit()**
- **ProDwgLayerItemInit()**
- **ProLayerItemAdd()**
- **ProLayerItemAddNoUpdate()**
- **ProLayerItemRemove()**
- **ProLayerItemRemoveNoUpdate()**

- **ProLayeritemLayersGet()**
- **ProLayerDisplaystatusGet()**
- **ProLayerDisplaystatusSet()**
- **ProLayerDisplaystatusNoUpdateSet()**
- **ProLayerDisplaystatusUpdate()**
- **ProDwgLayerDisplaystatusGet()**
- **ProDwgLayerDisplaystatusSet()**
- **ProLayerDisplaystatusSave()**
- **ProLayerDefLayerSet()**
- **ProLayerDefLayerGet()**
- **ProLayerViewDependencySet()**
- **ProLayerViewDependencyGet()**
- **ProLayerRuleExecute()**
- **ProLayerRuleCopy()**
- **ProLayerRuleMatch()**
- **ProLayeritemLayerStatusGet()**

To get the `ProLayer` object for a layer with the specified name and owner, call the function `ProMdlLayerGet()`. You must pass the name of the layer as a wide string.

To visit the layers in a model, use the function `ProMdlLayerVisit()`. As with other Creo Parametric TOOLKIT visit functions, you supply the visit action and visit filter functions.

The function `ProMdlLayersCollect()` collects a `ProArray` of layers in the model.

The function `ProLayerCreate()` creates a new layer with a specified name. It requires as input the `ProMdl` handle for the model that will own the layer. The function `ProLayerDelete()` deletes the layer identified by the specified `ProLayer` object.

The function `ProLayerItemsGet()` allocates and fills an array of `ProLayerItem` objects that contains the items assigned to the specified layer.

Note

The function `ProLayerItemsGet()` is deprecated. For a large number of layer items, the function `ProLayerItemsGet()` may return an error `PRO_TK_OUT_OF_MEMORY` to indicate that the function was unable to allocate a `ProArray` to hold all of the layer items. To address this issue, use the new function `ProLayerItemsPopulate()`.

The function `ProLayerItemsPopulate()` allocates and fills an array of `ProLayerItem` objects that contain the type and identifier of the items assigned to the specified layer. This function can retrieve a large number of items specified on the layer. Use the function `ProLayeritemarrayFree()` to free the allocated memory.

To initialize a `ProLayerItem`, call the function `ProLayerItemInit()`. This function should be used in all cases, except when all of the following are true:

- The layer owner is a drawing.
- The layer item owner is an assembly.
- The layer item is a component.
- You want to control the display status of this component only in a subassembly with a given path.

When all of the above conditions are true, use the function `ProDwgLayerItemInit()` to initialize the `ProLayerItem`.

To add items to a layer, call the function `ProLayerItemAdd()`, and pass as input a `ProLayer` object and the `ProLayeritem` object for the new layer item. To remove an item from a layer, use the function `ProLayerItemRemove()` and specify the `ProLayeritem` object for the item to remove.

The function `ProLayerItemAddNoUpdate()` adds the specified item to a layer without updating the model tree.

The function `ProLayerItemRemoveNoUpdate()` removes the specified item from the layer without updating the model tree.

To find all the layers containing a given layer item, use the function `ProLayeritemLayersGet()`. This function supports layers in solid models and in drawings.

As in an interactive session of Creo Parametric, one of the principal reasons to create a layer is to display or blank its member items selectively. The function `ProLayerDisplaystatusGet()` obtains the display status of the specified layer, in the form of the `ProLayerDisplay` enumerated type. The display status can be of following types:

- `PRO_LAYER_TYPE_NONE`—The selected layer is displayed. This is the default display status.
- `PRO_LAYER_TYPE_NORMAL`—The layer selected by the user is displayed.
- `PRO_LAYER_TYPE_DISPLAY`—The selected layer is isolated.
- `PRO_LAYER_TYPE_BLANK`—The selected layer is blanked.
- `PRO_LAYER_TYPE_HIDDEN`—The components in the hidden layers are blanked. This status is applicable only in the assembly mode.

To modify the display status of a layer, call the function `ProLayerDisplaystatusSet()`.

Note

`ProLayerDisplaystatusSet()` does not repaint the model after it modifies the display status. This is a temporary setting. It will be lost after you save or retrieve the model. To permanently change the display status, call the function `ProLayerDisplaystatusSave()`. However, the function `ProLayerDisplaystatusSet()` updates the model tree for the change in display status of the layer.

The function `ProLayerDisplaystatusNoUpdateSet()` sets the display status of a layer, without updating the model tree. It returns a boolean value `PRO_B_TRUE` for the output argument *is_update_tree_needed*, if the model tree requires an update for change in the display status of a layer. Use the function `ProLayerDisplaystatusUpdate()` to update the model tree for all the changes in the display statuses of all the layers in the specified model.

Unique functions are required to retrieve and set the status of layers in drawings. `ProDwgLayerDisplaystatusGet()` takes as input the layer handle and drawing view. The function `ProDwgLayerDisplaystatusSet()` takes an additional argument as input—the desired display status.

The function `ProLayerDisplaystatusSave()` saves the changes to the display status of all the layers in the specified owner. In addition, the display statuses are saved in the owner's submodels and drawing views.

To set up a default layer with a specified name, call the function `ProLayerDefLayerSet()`. This function requires the default layer type, which is defined in the enumerated type `ProDefLayerType`. To get the name of the default layer with the specified type, call the function `ProLayerDefLayerGet()`.

The function `ProLayerViewDependencySet()` sets the display of layers of the specified view to depend on the display of layers in the drawing. The syntax of this function is as follows:

```
ProLayerViewDependencySet (

  ProView      view,
  ProBoolean   depend);
```

If *depend* is set to `PRO_B_TRUE`, the layers in the view will be displayed when the layers in the drawing are displayed. If *depend* is set to `PRO_B_FALSE`, the layer display in the view will be independent of the display in the drawing. To determine whether the layer display in the view is dependent on the display in the drawing, call the function `ProLayerViewDependencyGet()`.

You can define rules in layers. Use the function `ProLayerRuleExecute()` to execute the layer rules on the specified model. The rules must be enabled in the layers to be executed.

The function `ProLayerRuleCopy()` copies the rules from the reference model to the current model for the specified layer. The input arguments are:

- `CurrentModel`—Specifies the current model to which the layer rules must be copied.
- `LayerName`—Specifies the name of an existing layer in both the models. To copy the layer rules, the name of the layer `LayerName` in both the models must be the same.
- `ReferenceModel`—Specifies the reference model from which the layer rules must be copied.

Use the function `ProLayerRuleMatch()` to compare the rules between the current and reference model for the specified layer. The name of the layer `LayerName` in both the models must be the same, for comparing the layer rules.

The function `ProLayeritemLayerStatusGet()` returns the status of an item for the specified layer. The input arguments are:

- *pro_drawing*—Specifies the drawing which is the owner of the layer that contains the specified item.
- *pro_layer_item*—Specifies the layer item. If the owner type of the layer item is `PRO_LAYITEM_FROM_PATH`, it is mandatory to specify the *pro_drawing* input argument.
- *pro_layer*—Specifies the layer that contains the item.

The output argument returns the status of the item using the enumerated data type `ProLayerItemStatus`. The valid values are:

- `PRO_LAY_ITEM_STATUS_INCLUDE`—Specifies that the status of the layer item is **Include**. The item is included in the specified layer.
- `PRO_LAY_ITEM_STATUS_EXCLUDE`—Specifies that the status of the layer item is **Exclude**. The item is excluded in the specified layer.
- `PRO_LAY_ITEM_STATUS_ADDED_BY_RULE`—Specifies that the status of the layer item is defined by rules.

The enumerated data type `ProLayerItemStatus` must have one of the following values:

- `PRO_LAY_ITEM_STATUS_INCLUDE`
- `PRO_LAY_ITEM_STATUS_EXCLUDE`
- `PRO_LAY_ITEM_STATUS_INCLUDE` and `PRO_LAY_ITEM_STATUS_ADDED_BY_RULE`
- `PRO_LAY_ITEM_STATUS_EXCLUDE` and `PRO_LAY_ITEM_STATUS_ADDED_BY_RULE`

Example 3: Creating a Layer

The sample code in `UgModelLayerCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_model` shows how to create a layer and add items to it.

This example streamlines the layer creation process (compared to interactive Creo Parametric) because the application creates the layer and adds items to it in only one step. Note that this example does not allow users to add a subassembly to the new layer.

Notebook

The functions described in this section work with notebook (`.lay`) files.

Functions Introduced:

- **ProLayoutDeclare()**
- **ProLayoutUndeclare()**
- **ProLayoutRegenerate()**

The function ProLayoutDeclare() declares a notebook name to the specified Creo Parametric model or notebook. You can resolve conflicts using the enumerated type ProDeclareOptions. It has the following values:

- PRO_DECLARE_INTERACTIVE—Resolves the conflict in interactive mode.
- PRO_DECLARE_OBJECT_SYMBOLS—Keep the symbols in the specified Creo Parametric model or notebook object.
- PRO_DECLARE_LAYOUT_SYMBOLS—Keep the symbols specified in the notebook.
- PRO_DECLARE_ABORT—Abort the notebook declaration process and return an error.

Use the function ProLayoutUndeclare() to undeclare the notebook name to the specified Creo Parametric model or notebook. You can resolve conflicts using the enumerated type ProUndeclareOptions. It has the following values:

- PRO_UNDECLARE_FORCE—Continues to undeclare the notebook even if references exist.
- PRO_UNDECLARE_CANCEL—Does not undeclare the notebook if references exist.

The method ProLayoutRegenerate() regenerates the specified notebook.

Visiting Displayed Entities

Functions Introduced:

- **ProSolidDispCompVisit()**
- **ProAsmcomppathDispPointVisit()**
- **ProAsmcomppathDispCurveVisit()**
- **ProAsmcomppathDispCsysVisit()**
- **ProAsmcomppathDispQuiltVisit()**

The functions in this section enable you to find quickly all the entities (points, datum curves, coordinate systems, and quilts) currently displayed in an assembly. It is possible to do this using the regular Creo Parametric TOOLKIT functions for visiting assembly components and entities, together with the ProLayer functions explained earlier in this chapter; but the functions described here are much more efficient because they make use of Creo Parametric's internal knowledge of the display structures.

The function ProSolidDispCompVisit() traverses the components at all levels in an assembly which are not blanked by a layer. The visit action function is called on both the downward traversal and the upward one, and is given a boolean input to distinguish them. It is also given the assembly path and the solid handle to

the current subassembly. The subassembly could be found from the path using `ProAsmcomppathMdlGet()`, of course, but Creo Parametric passes this to the action function to allow greater efficiency.

The functions `ProAsmcomppathDisp*Visit()` visit the entities in a subassembly that are not blanked by a layer at any level in the root assembly.

3

Core: Solids, Parts, and Materials

This chapter describes how to access solids, parts, and their contents.

Solid Objects

The Creo Parametric term “solid” denotes a part or an assembly. The object is called `ProSolid` and is declared as an opaque handle. It is an instance of `ProMdl` and can be cast to that type to use functions that have the prefix “`ProMdl`”.

Creating a Solid

Function Introduced:

- **ProSolidMdlnameCreate()**

The function `ProSolidMdlnameCreate()` creates an empty part or assembly with the specified name, and provides a handle to the new object. It does not make the new solid current, nor does it display the solid. In Creo Parametric 7.0.0.0 and later, an empty part is created with absolute accuracy, by default. Refer to the Creo Parametric help for more information on Model Accuracy.

Contents of a Solid

Functions Introduced:

- **ProSolidFeatVisit()**
- **ProSolidQuiltVisit()**
- **ProSolidAxisVisit()**
- **ProSolidCsysVisit()**
- **ProSolidFeatstatusGet()**
- **ProSolidFeatstatusSet()**
- **ProSolidFeatstatusWithoptionsSet()**
- **ProSolidFeatstatusflagsGet()**
- **ProSolidFailedFeatsList()**
- **ProSolidFailedfeaturesList()**
- **ProSldsurfaceShellsAndVoidsFind()**
- **ProSolidToleranceStandardGet()**
- **ProSolidToleranceStandardSet()**

The following visit functions enable you to access the various types of objects inside a part or assembly:

- `ProSolidFeatVisit()`—Visits all the features, including those used internally (which are not visible to the Creo Parametric user). You can also use this function to visit the components of an assembly.
- `ProSolidSurfaceVisit()`— Visits the surfaces of the model only if the model has a single body else returns the error `PRO_TK_MULTIBODY_UNSUPPORTED`. This includes all surfaces created by solid features, but not datum surfaces.
- `ProSolidQuiltVisit()`—Visits all the quilts in a part or an assembly.
- `ProSolidAxisVisit()`—Visits all the axes in a part or an assembly.
- `ProSolidCsysVisit()`—Visits all the coordinate system datums in a part or an assembly.

The function `ProSolidFeatstatusGet()` retrieves a list of the integer identifiers and statuses of all the features in a specified solid in the order in which they are regenerated. The integer identifier of a feature is the value of the `id` field in the `ProFeature` object and also the INTERNAL ID seen in Creo Parametric.

The function `ProSolidFeatstatusSet()` enables you to set the regeneration order and statuses of the features in the solid.

The function `ProSolidFeatstatusWithoptionsSet()` assigns the regeneration order and status bit flags for the specified features in a solid based on the bitmask containing one or more regeneration control bit flags of the type `PRO_REGEN_*` defined in `ProSolid.h`. Refer to the Regenerating a Solid on page 96 section for more information on the bit flags.

The function `ProSolidFeatstatusflagsGet()` retrieves the array of integer identifiers of the features in a specified solid and the corresponding array of bitmasks representing one or more feature status bit flags of the type `PRO_FEAT_STAT_*` defined in `ProFeature.h`. Refer to the Core: Features on page 131 chapter for more information on the feature status bit flags.

The function `ProSolidFailedFeatsList()` retrieves the list of identifiers of failed features in a specified solid.

Note

From Pro/ENGINEER Wildfire 5.0 onward, the function `ProSolidFailedFeatsList()` has been deprecated. Use the function `ProSolidFailedfeaturesList()` instead. Pass `NULL` for the input arguments *co_failed_ids* and *co_x_failed_ids* while using `ProSolidFailedfeaturesList()` in the Resolve mode.

The function ProSolidFailedfeaturesList() retrieves the list of identifiers of all or any of the failed features, children of failed features, children of external failed features, or both the features and their children.

The function ProSldsurfaceShellsAndVoidsFind() returns an ordered list of surface-contour pairs for each shell and void in a solid.

The surface-contour pairs describe the shell faces and are specified by the ProSldsurfaceShellface objects. If the contour field in the ProSldsurfaceShellface object is NULL, it means all the contours in the geometry belong to the same surface and the shell_faces array contains only one surface ID. However, if the contours belong to different shells, the shell_faces array contains items equal to the number of contours.

The ordered list of surface-contour pairs is specified by the ProSldsurfaceShellorder objects; each of this object contains the following fields:

- orientation—Specifies the shell orientation. If this field is 1, the shell is oriented outward, if it is -1, the shell is inward oriented meaning it is a void.
- first_face—Specifies the index in the array of ProSldsurfaceShellface objects.
- number_of_faces—Specifies the total number of shell faces.
- ambient_shell—Specifies the index in the array of ProSldsurfaceShellorder objects.

The function ProSolidToleranceStandardGet() returns the tolerance standard assigned to a solid. Use the method ProSolidToleranceStandardSet() to set the tolerance standard for a solid. After you set the tolerance standard for a solid, you must regenerate the solid, if required. The function does not regenerate the solid.

Displaying a Solid

Function Introduced:

- **ProSolidDisplay()**

The function ProSolidDisplay() displays a solid in the current Creo Parametric window. This does not make the object current from the point of Creo Parametric.

Example 1: Loading and Displaying a Solid

The example in the file UgSolidLoadDisp.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_solid, shows how to use the functions ProObjectwindowMdlnameCreate() and ProSolidDisplay().

Regenerating a Solid

Function Introduced:

- **ProSolidRegenerate()**
- **ProSolidRegenerationIsNoresolvemode()**
- **ProSolidRegenerationstatusGet()**

The function `ProSolidRegenerate()` regenerates the specified solid. One of the inputs to the function is a bitmask that specifies how the regeneration must be performed. The bitmask may contain the following flags:

- `PRO_REGEN_NO_FLAGS`—Equivalent to passing no flags.
- `PRO_REGEN_CAN_FIX`—Allows the user to interactively fix the model using the user interface, if regeneration fails. This bit flag needs to be set only in case of interactive applications. If this option is not included, the user interface does not update even if regeneration is successful. Use `ProWindowRepaint()` and `ProTreetoolRefresh()` to perform the update if needed. Also, this bit flag must be set only in the Resolve mode. Otherwise, `ProSolidRegenerate()` returns `PRO_TK_BAD_CONTEXT`.
- `PRO_REGEN_ALLOW_CONFIRM`—This flag has been deprecated from Creo Parametric 4.0 M030. Allows the user to interactively select the option of retaining failed features and children of failed features via a pop-up dialog box, if regeneration fails. This bit flag must be set only in the No-Resolve mode. Otherwise, `ProSolidRegenerate()` returns `PRO_TK_BAD_CONTEXT`.

> **Note**
>
> The interactive dialog box which provided an option to retain failed features and children of failed features, if regeneration fails is no longer supported. Creo Parametric displays a warning message which gives details of failed features.

- `PRO_REGEN_UNDO_IF_FAIL`—Allows the user to undo the failed regeneration and restore the previous status. This flag needs to be set only in the No-Resolve mode. Otherwise, `ProSolidRegenerate()` returns `PRO_TK_BAD_CONTEXT`. The result obtained may be different from the one attained by using the **Restore** option in the Resolve mode. **Restore** in the Resolve mode can be used immediately after the first failure. But undo in the No-Resolve mode due to this bit flag happens only after all the features are regenerated or failed. In some cases, the undo may not happen at all.

Note

The bit flags PRO_REGEN_ALLOW_CONFIRM and PRO_REGEN_UNDO_IF_FAIL are not compatible with each other. Setting both of them together will result in PRO_TK_BAD_CONTEXT.

- PRO_REGEN_SKIP_DISALLOW_SYS_RECOVER—Skips the preparation for failure recovery. If this option is used, Undo Changes is possible if a failure occurs. This option is used only in conjunction with PRO_REGEN_CAN_FIX.
- PRO_REGEN_UPDATE_INSTS—Updates instances of the solid in memory. This may slow down the regeneration process.
- PRO_REGEN_RGN_BCK_USING_DISK—Stores the backup model on the disk. This is useful only if PRO_REGEN_CAN_FIX is set.
- PRO_REGEN_FORCE_REGEN—Forces the solid to fully regenerate. This will regenerate every feature in the solid. If not set, Creo Parametric uses its internal algorithm to determine which features to regenerate.
- PRO_REGEN_TOP_ASM_ONLY—Forces only top level assembly to regenerate. This flag forces the regeneration of all the features and components that are defined in the specified top level assembly, even when they are considered up-to-date. The features and components from the low level assembly that are essential for the correct assembly regeneration results, might also be regenerated during the regeneration process. However, they are not excluded from the regeneration process.

Note

This flag cannot be used with PRO_REGEN_FORCE_REGEN.

- PRO_REGEN_UPDATE_ASSEMBLY_ONLY—Updates assembly and sub-assembly placements and regenerates assembly features and intersected parts. If the affected assembly is retrieved as a simplified representation, this flag will update the locations of the components. If the flag is not set, the component locations are not updated by default when the simplified representation is retrieved.

Note

This flag cannot be used with `PRO_REGEN_FORCE_REGEN`.

- `PRO_REGEN_RESUME_EXCL_COMPS`—Enables Creo Parametric to resume available excluded components of the simplified representation during regeneration. This can result in a more accurate update of the simplified representation.

Note

Component models which are not in session at the time of the call to `ProSolidRegenerate()` will not be retrieved due to this option.

- `PRO_REGEN_NO_RESOLVE_MODE` — Specifies the No-Resolve mode introduced in Pro/ENGINEER Wildfire 5.0. This is the default mode in Creo Parametric. In this mode, if a model and feature regeneration fails, failed features and children of failed features are created and regeneration of other features continues.
- `PRO_REGEN_RESOLVE_MODE` — Specifies the Resolve mode. In this mode, you can continue with the Pro/ENGINEER Wildfire 4.0 behavior, wherein if a model and feature regeneration fails, the failure needs to be resolved before regeneration can be resumed. You can also switch to the Resolve mode by setting the configuration option `regen_failure_handling` to `resolve_mode` in the Creo Parametric session.

  In Creo Parametric 7.0.1.0 and later, the configuration option `regen_failure_handling` has been deprecated. If a model and feature regeneration fails and if you want to use Resolve mode, you need to contact PTC Customer Support. For more information, refer to the section Contacting PTC Technical Support in the *Getting Started with Creo Parametric TOOLKIT* guide.

Note

Setting the configuration option to switch to Resolve mode ensures the old behavior as long as you do not retrieve the models saved under the No-Resolve mode. To consistently preserve the old behavior, use Resolve mode from the beginning and throughout your Creo Parametric session. Temporarily setting the bit flag `PRO_REGEN_RESOLVE_MODE` in the relevant functions does not ensure the old behavior.

The function `ProSolidRegenerationIsNoresolvemode()` identifies if the regeneration mode in the active Creo Parametric session is the No-Resolve mode. Set the `ProBoolean` argument *is_no_resolve* to `PRO_B_TRUE` to set the No-Resolve mode.

The function `ProSolidRegenerationstatusGet()` returns the regeneration status of the solid model. This status is similar to the regeneration status indicator (Green/Yellow/Red) in the Creo Parametric User Interface.

The regeneration status can take one of the following values:

- `PRO_SOLID_REGENERATED`—Specifies that the model is up-to-date and requires no regeneration.
- `PRO_SOLID_NEEDS_REGENERATION`—Specifies that the model has changed and requires regeneration.
- `PRO_SOLID_FAILED_REGENERATION`—Specifies that the regeneration has failed or has warnings.

Note

Models with certain contents, such as circular references or assembly analysis features, will never return a fully “regenerated” status. Thus, this status should not provide an absolute restriction. If the flag remains in the “`PRO_SOLID_NEEDS_REGENERATION`” status through two successful regenerations, the model could be considered up-to-date.

- `PRO_SOLID_CONNECT_FAILED`—Specifies that the model has successfully regenerated, however, the connect operation for mechanisms in the solid model has failed. This status is applicable only for assemblies which have moving components.

Example 2: Combining Regeneration Flags

The sample code in `UgSolidRegen.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_solid` shows how to use the function `ProSolidRegenerate()`.

Combined States of a Solid

With a combined state, you can combine and apply multiple display states to a Creo Parametric model. Combined states are composed of the following two or more display states:

- Saved Views
- Layer state
- Annotations
- Cross section
- Exploded view
- Simplified representation
- Model style

The object `ProCombstate` for a combined state has the same declaration as the `ProModelitem` object, only with the type set to `PRO_COMBINED_STATE`. The declaration is as follows:

```
typedef struct pro_model_item
{
  ProType  type;
  int      id;
  ProMdl   owner;
}ProCombstate;
```

Functions Introduced:

- **ProMdlCombstatesGet()**
- **ProCombstateActiveGet()**
- **ProCombstateDataGet()**
- **ProCombstateActivate()**
- **ProMdlCombStateCreate()**
- **ProCombstateRedefine()**
- **ProCombstateAnnotationsGet()**
- **ProCombstateAnnotationsAdd()**
- **ProCombstateAnnotationsRemove()**
- **ProCombstateDelete()**
- **ProCombstateAnnotationsStateGet()**

- **ProCombstateSupplGeomStateGet()**
- **ProCombstateAnnotationsAndSupplGeomStateSet()**
- **ProCombstateIsPublished()**
- **ProCombstateIsDefault()**

The function `ProMdlCombstatesGet()` returns an array of combined states of a specified solid.

The function `ProCombstateActiveGet()` retrieves the active combined state in a specified solid model. The active combined state is the default state when the model is opened.

The function `ProCombstateDataGet()` returns information for a specified combined state. The output arguments of this function are:

- *cs_name*—The name of the combined state.
- *cs_ref_arr*—An array of reference states of the type `ProModelitem`. This array can contain states of the following types:
  - `PRO_VIEW`
  - `PRO_LAYER_STATE`
  - `PRO_SIMP_REP`
  - `PRO_EXPLD_STATE`
  - `PRO_XSEC`
  - `PRO_STYLE_STATE`
- *p_clip_opt*—A pointer to the value of the cross section clip. This is applicable only in case of a valid reference of the type `PRO_XSEC`. The `PRO_XSEC` item represents a `ProXsec` object or a zone feature.

  The values for the cross section clip are specified by the enumerated type `ProCrossecClipOpt`. The possible values are as follows:
  - `PRO_VIS_OPT_NONE`—Specifies that the cross section or zone feature is not clipped.
  - `PRO_VIS_OPT_FRONT`—Specifies that the cross section or zone feature is clipped by removing the material on the front side. The front side is where the positive normals of the planes of the cross section or zone feature are directed.
  - `PRO_VIS_OPT_BACK`—Specifies that the cross section or zone feature is clipped by removing the material on the back side.
- *p_is_expld*—A `ProBoolean` value that specifies whether the combined state is exploded. This value is available only if when a valid `PRO_EXPLD_STATE`

reference state is retrieved. It is not available for Creo Parametric parts since an exploded state does not exist in the part mode.

Use the function `ProCombstateActivate()` to activate a specified combined state.

The function `ProMdlCombStateCreate()` creates a new combined state based on specified references. The input arguments of this function are as follows:

- *p_solid*—Specify the solid model in which you want to create a new combined state.
- *new_name*—Specify the name of the new combined state.
- *ref_arr*—Specify the array of reference states. Refer to the description of the argument *cs_ref_arr* on page of the function `ProCombstateDataGet()` for the valid reference states.
- *clip_opt*—Specify the value of the cross section clip. Refer to the description of the argument *p_clip_opt* on page of the function `ProCombstateDataGet()` for more information.
- *is_expld*—Specify `PRO_B_TRUE` if the combined state is exploded, else specify `PRO_B_FALSE`. This argument needs to be set only in case of a valid `PRO_EXPLD_STATE` reference state. It is not applicable for Creo Parametric parts since an exploded state does not exist in the part mode.

Use the function `ProCombstateRedefine()` to redefine a created combined state. The values specified by the input arguments *ref_arr*, *clip_opt*, and *is_expld* of the function `ProMdlCombStateCreate()` are redefined.

In case you do not want to redefine a reference state, pass the reference state with the same value. While redefining, you must specify reference states. If you do not pass reference states, the combined state is redefined to a `NO_STATE` state. `NO_STATE` state means the display of the reference state is not changed on activation of combined state.

In case you want to refine or create a combined state that uses the most recently used instance of a reference state, use the `PRO_COMBSTATE_REF_MRU` option as the id field of that type of reference state in the `ProModelitem` object.

The function `ProCombstateAnnotationsGet()` retrieves an array of annotations and their status flags from a specified combined state item.

The function `ProCombstateAnnotationsAdd()` adds an array of annotations to a specified combined state item. The input argument *status_flags* specifies if an annotation must be displayed in the combined state. If you specify the value `0` for this argument, then the annotation is displayed in the combined state. If you specify the value `1`, then the annotation is not displayed.

Use the function `ProCombstateAnnotationsRemove()` to remove the annotations from a specified combined state item.

Use the function `ProCombstateDelete()` to delete a specified combined state. The function fails if the specified combined state is the default or active combined state.

Annotations and annotation elements can be assigned to a combined state. When the combined state is active, the annotations are displayed in the graphics window. Similarly, annotations and annotation elements can be assigned to layers. Supplementary geometry such as datum planes, points, coordinate systems, axes, curves, and surfaces can also be assigned to combined state or layers. The display of annotations and supplementary geometry in a model is controlled either by the combined state or layers. The configuration option `combined_state_type` is used to define how the visibility of annotations and supplementary geometry is controlled in a new combined state.

Refer to Creo Parametric Help for more information.

The function `ProCombstateAnnotationsStateGet()` checks if the display of annotations is controlled by the specified combined state or layers. The function returns `PRO_B_TRUE` when the display is controlled by combined state. Use the function `ProCombstateSupplGeomStateGet()` to check if the display of supplementary geometry is controlled by the specified combined state or layers. The function returns `PRO_B_TRUE` when the display is controlled by combined state.

The function `ProCombstateAnnotationsAndSupplGeomStateSet()` allows you to change the display of annotations and supplementary geometry by the combined state or layers. The input arguments follow:

- *cs_item*—Specifies a pointer to the combined state item from which the state needs to be retrieved.
- *annotation_state*—Flag to set the specified combined state to annotation state. Pass the value of *annotation_state* as `PRO_B_TRUE` if display of annotations should be controlled by combined state. Pass it as `PRO_B_FALSE` if display of annotations should be controlled by layers.
- *supplgeom_state*—Flag to set the specified combined state to supplementary geometry state. Pass the value of *supplgeom_state* as `PRO_B_TRUE` if display of supplementary geometry should be controlled by combined state. Pass it as `PRO_B_FALSE` if display of supplementary geometry should be controlled by layers.

Note

If value of the input argument *supplgeom_state* is set to `PRO_B_TRUE`, the value of *annotation_state* also, must be set to `PRO_B_TRUE`.

The function returns PRO_TK_NO_CHANGE if the current states of annotations and supplementary geometry in the combined state are same as the requested states.

The function ProCombstateIsPublished() checks if the specified combined state has been published to Creo View.

Use the function ProCombstateIsDefault() checks if the specified combined state is set as the default combined state for the model.

Layer State

A layer state stores the display state of existing layers and all the hidden layers of the top-level assembly. You can create and save one or more layer states and switch between them to change the assembly display.

The object ProLayerstate represents the layer state. It has the same declaration as the ProModelitem object, only with the type set to PRO_LAYER_STATE. The declaration is as follows:

```
typedef struct pro_model_item
{
  ProType  type;
  int      id;
  ProMdl   owner;
}ProLayerstate;
```

Functions Introduced:

- **ProLayerstatesGet()**
- **ProLayerstateActiveGet()**
- **ProLayerstateNameGet()**
- **ProLayerstateActivate()**
- **ProLayerstateCreate()**
- **ProLayerstateLayersGet()**
- **ProLayerstateLayerAdd()**
- **ProLayerstateLayerRemove()**
- **ProLayerstateActivestateUpdate()**
- **ProLayerstateModelitemHide()**
- **ProLayerstateModelitemUnhide()**
- **ProLayerstateModelitemIsHidden()**
- **ProLayerstateDelete()**

The function ProLayerstatesGet() returns an array of layer states for a specified solid.

The function `ProLayerstateActiveGet()` retrieves the active layer state in a specified solid model.

The function `ProLayerstateNameGet()` retrieves the name of a specified layer state.

Use the function `ProLayerstateActivate()` to activate a specified layer state.

The function `ProLayerstateCreate()` creates a new layer state based on specified references. The input arguments of this function are as follows:

- *p_solid*—Specify the solid model in which you want to create a new layer state.
- *state_name*—Specify the name of the new layer state. The name can only consist of alphanumeric, underscore, and hypen characters.
- *layers*—Specify an array of reference layers.
- *disp_arr*—Specify an array of display statuses. The number of display statues is equal to the number of reference layers.
- *hidden_items*—Specify an array of hidden items.

> **Note**
>
> `ProLayerItem` of type `PRO_LAYER_LAYER` is not supported in the function `ProLayerstateCreate()`, when you create a new layer state.

The function `ProLayerstateLayersGet()` retrieves the reference data for a specified layer state.

The function `ProLayerstateLayerAdd()` adds a new layer to an existing layer state. Specify the new layer, its display state, and the name of the existing layer state as input arguments to this function.

The function `ProLayerstateLayerRemove()` removes a specific layer from a specified layer state.

The function `ProLayerstateActivestateUpdate()` updates the layer state, which is active in the specified model. If the display statues of layers have changed, then calling this function ensures that the active layer state in the model is updated with the new display statuses of the layers.

Use the function `ProLayerstateModelitemHide()` to hide the display of a specific item on the specified layer state.

Use the function `ProLayerstateModelitemUnhide()` to remove a specific item from the list of hidden items on a layer state.

Use the function `ProLayerstateModelitemIsHidden()` to identify if an item is hidden on a layer state.

Use the function `ProLayerstateDelete()` to delete a specified layer state.

Evaluating Mathematical Expressions for a Solid

Functions Introduced:

- **ProMathExpressionEvaluate()**

The function `ProMathExpressionEvaluate()` evaluates the given mathematical expression in the context of a given solid. The expression may include parameters, dimensions, embedded functions, or predefined constants. This function returns a pointer to the calculated result and a pointer to the unit of the calculated result.

Solid Outline

Functions Introduced:

- **ProSolidOutlineGet()**
- **ProSolidOutlineCompute()**

The function `ProSolidOutlineGet()` provides you with the maximum and minimum values of X, Y, and Z occupied by the contents of the solid, with respect to the default, solid coordinate system.

The function `ProSolidOutlineCompute()` calculates the outline of the solid with respect to any orientation, defined by a transformation matrix. (For more information, see the Core: Coordinate Systems and Transformations on page 222 chapter.) The function enables you to exclude from the calculation items of any or all of the following types:

- Datum plane
- Datum point
- Datum axes
- Datum coordinate system
- Facets

Example 3: Computing the Outline of a Solid

The sample code in `UgSolidOutlineComp.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_solid` computes the outline of a solid with respect to a selected coordinate system, and converts the result back to solid coordinates.

Solid Accuracy

Functions Introduced:

- **ProSolidAccuracyGet()**
- **ProSolidAccuracySet()**
- **ProSolidMaxsizeGet()**

Use the functions `ProSolidAccuracyGet()` and `ProSolidAccuracySet()` to retrieve and set the accuracy of a specified part or assembly, respectively.

Note

To set or retrieve the accuracy for an assembly you must traverse through all its parts in the assembly with these functions.

The input arguments for the function `ProSolidAccuracySet()` are as follows:

- *solid*—The part or assembly whose accuracy you want to set.
- *type*—The type of the accuracy. The valid values are:
  - `PRO_ACCURACY_RELATIVE`—Specifies the relative accuracy
  - `PRO_ACCURACY_ABSOLUTE`—Specifies the absolute accuracy
- *accuracy*—The value of the accuracy that you want to set. The unit used for the absolute accuracy of the dimension is based on the unit of the part or assembly.

Note

Regenerate the model using the function `ProSolidRegenerate()` after setting the accuracy using `ProSolidAccuracySet()`.

The function `ProSolidAccuracyGet()` returns the type and value of the accuracy. The accuracy may be relative or absolute.

Using these functions to set and retrieve part accuracy is similar to performing these functions in Creo Parametric using **File ► Prepare ► Model Properties**.

Derive the geometry epsilon for the required relative accuracy as follows:

```
geometry_epsilon = max_model_size  x relative_accuracy x 0.08333
```

where, `max_model_size` is the output returned by the function `ProSolidMaxsizeGet()` and `0.08333` is the scaling factor.

Use the function `ProSolidMaxsizeGet()` to get the maximum model size of the specified solid. The maximum model size does not decrease even when material is removed from the solid.

Solid Units

Introduction to Unit of Measurement and System of Units

Each model has a basic system of unit to ensure that all material properties of that model are consistently measured and defined. All models are defined on the basis of system of units. A part can have only one system of unit.

Following are the types of quantities which govern the definition of unit of measurement:

- Basic Quantities—The basic units and dimensions of the system of units. For example, consider the Centimeter Gram Second (CGS) system of unit. The basic quantity for this system of unit is:
  - Length—cm
  - Mass—g
  - Force—dyne
  - Time—sec
  - Temperature—K
- Derived Quantities—The derived units are those that are derived from the basic quantities. For example, consider the Centimeter Gram Second (CGS) system of unit. The derived quantities for this system of unit are as follows:
  - Area—cm^2
  - Volume—cm^3
  - Velocity—cm/sec

Types of Systems of Units

Following are the types of system of units:

- Pre-defined system of unit—This system of unit is provided by default.
- Custom-defined system of unit—This system of unit is defined by the user only if the model does not contain standard metric or nonmetric units or if the material file contains units that cannot be derived from the predefined system of units or both.

In Creo Parametric, the system of units are categorized as follows:

- Mass Length Time (MLT)

- The following systems of units belong to this category:
  - CGS —Centimeter Gram Second
  - MKS—Meter Kilogram Second
  - mmKS—millimeter Kilogram Second
- Force Length Time (FLT)
  - The following systems of units belong to this category:
  - Creo Parametric Default—Inch lbm Second. This is the default system followed by Creo Parametric.
  - FPS—Foot Pound Second
  - IPS—Inch Pound Second
  - mmNS—Millimeter Newton Second

Definitions

For Creo Parametric TOOLKIT, a system of units is represented by the structure `ProUnitsystem`. This structure is defined as:

```
typedef struct {
  ProMdl  owner;
  ProName name;
}ProUnitsystem;
```

where the name is the user-visible name used in the Unit Manager dialog.

An individual unit is represented by the structure `ProUnititem`. This structure is defined as:

```
typedef struct {
  ProMdl  owner;
  ProName name;
}ProUnititem;
```

where the name is the user-visible abbreviation used in the Unit Manager dialog.

Note

The functions described in the following sections supersede the functions `prodb_get_model_units()` and `prodb_set_model_units()`.

Retrieving Systems of Units

Functions Introduced:

- **ProMdlUnitsystemsCollect()**
- **ProMdlPrincipalunitsystemGet()**

Use the function `ProMdlUnitsystemsCollect()` to retrieve the set of systems of units which are accessible to the model in the form of an array. The input arguments of the function are as follows:

- *mdl*—Specifies a handle to the model.

The function outputs a `ProArray` containing the set of systems of units for the specified model.

Note

The function retrieves both the pre-defined as well as the custom-defined system of unit.

Use the function `ProMdlPrincipalunitsystemGet()` to retrieve the principal system of units for the specified model.

Modifying Systems of Units

Functions Introduced:

- **ProUnitsystemRename()**
- **ProUnitsystemDelete()**

Use the function `ProUnitsystemRename()` to rename a custom- defined system of unit. The input parameters for this function are as follows:

- *system*—Specifies a handle to the system of unit.
- *new_name*—Specifies the new name for the system.

Use the function `ProUnitsystemDelete()` to delete a custom- defined system of unit. Specify a handle to the system of units to be deleted as the input parameter for this function.

Note

You can only delete a custom-defined system of units. You cannot delete a pre-defined system of units.

Accessing Systems of Units

Functions Introduced:

- **ProUnitsystemIsStandard()**
- **ProUnitsystemTypeGet()**
- **ProUnitsystemUnitGet()**

Use the function `ProUnitsystemIsStandard()` to check whether the system of unit is a Creo Parametric standard system. Specify the name of the system of unit as the input parameter.

Use the function `ProUnitsystemTypeGet()` to retrieve the type of system of unit. Specify the name of the system of unit as the input argument. The output argument of this function is as follows:

- *type*—The type of system of unit. It can have the following values:
  - `PRO_UNITSYSTEM_MLT`—Mass Length Time
  - `PRO_UNITSYSTEM_FLT`—Force Length Time

For more information on the same refer to the section on Types of Systems of Units on page 108 above.

Use the function `ProUnitsystemUnitGet()` to retrieve the unit of particular type for a specified system of unit.

Creating a New System of Units

Function Introduced:

- **ProMdlUnitsystemCreate()**

Use the function `ProMdlUnitsystemCreate()` to create a new system of unit or to create a copy of an existing system of unit. The function expects the following input parameters:

- *mdl*—Specifies a handle to the model.
- *type*—Specifies the new type of system of unit.
- *name*—Specifies the name of the new system of unit.
- *units*—Specifies the set of units for the new system of unit created.

It outputs the newly created system of unit.

Accessing Individual Units

Functions Introduced:

- **ProMdlUnitsCollect()**
- **ProUnitInit()**
- **ProUnitTypeGet()**
- **ProUnitNameGet()**

- **ProUnitConversionGet()**
- **ProUnitConversionCalculate()**
- **ProUnitIsStandard()**
- **ProUnitExpressionGet()**
- **ProUnitInitByExpression()**
- **ProUnitCreateByExpression()**
- **ProUnitModifyByExpression()**

Use the function `ProMdlUnitsCollect()` to retrieve a set of units of a particular type that are available to the specified model.

The function `ProUnitInit()` retrieves the unit of a particular name for a specified model.

Note

The function is applicable only for basic units and not for derived ones.

The function `ProUnitNameGet()` returns the name of the unit. For system generated unit, that has no user-friendly name, it returns the error `PRO_TK_NOT_DISPLAYED`.

Use the function `ProUnitTypeGet()` to retrieve the unit type of a particular unit.

Unit types can have any of the following values:

- `PRO_UNITTYPE_LENGTH`
- `PRO_UNITTYPE_MASS`
- `PRO_UNITTYPE_FORCE`
- `PRO_UNITTYPE_TIME`
- `PRO_UNITTYPE_TEMPERATURE`
- `PRO_UNITTYPE_ANGLE`

Use the function `ProUnitConversionGet()` to retrieve the conversion factor for a particular unit. The output arguments of this function are:

- *conversion*—Specifies the conversion factor for a unit in terms of scale of the unit and an offset value.

```
Example - Consider the formula to convert temperature from Centigrade
 to Fahrenheit
F = a + (C * b)
where
F is the temperature in Fahrenheit
C is the temperature in Centigrade
```

```
a = 32 (constant signifying the offset value)
b = 9/5 (ratio signifying the scale of the unit)
```

Note

Creo Parametric scales the length dimensions of the model using the factor specified. If the scale is modified, the model is regenerated. When you scale the model, the model units are not changed. Imported geometry cannot be scaled.

- *ref_unit*— Specifies the reference unit for the conversion.

Use the function `ProUnitConversionCalculate()` to calculate the conversion factor between two units. These units can belong to the same model or two different models.

Use the function `ProUnitIsStandard()` to determine whether the unit is a standard unit as defined in Creo Parametric.

Creo Parametric uses named quantities to represent units other than the basic units (e.g. "Ilbs_stress_unit", which represents a quantity of stress in the default Creo Parametric unit systems). Parameters and material properties which are assigned derived units will return the name in the `ProUnititem` structure, rather than the actual unit-based expression for the quantity.

Use the function `ProUnitExpressionGet()` to retrieve the unit-based expression for a given Creo Parametric unit name.

Use the function `ProUnitInitByExpression()` to retrieve the `ProUnititem` given a unit-based expression.

The function `ProUnitCreateByExpression()` creates a derived or basic unit, based on expression. Use the function `ProUnitModifyByExpression()` to modify a derived unit.

Modifying Units

Functions Introduced:

- **ProUnitModify()**
- **ProUnitDelete()**
- **ProUnitRename()**

Use the function `ProUnitModify()` to modify a pre-defined unit. Modifying the units can invalidate your relations, as they are not scaled along with the model. The input parameters are:

- *unit*— Specifies the unit to be modified.
- *conversion*—Specifies the conversion factor for the unit.
- *ref_unit*—Specifies the reference unit.

Use the function `ProUnitDelete()` to delete a previously created unit.

Note

You cannot delete a pre-defined unit. If you delete a unit, you cannot undo the deletion.

Use the function `ProUnitRename()` to rename an existing unit.

Creation of a new Unit

- **ProUnitCreate()**

Use the function `ProUnitCreate()` to create a new basic unit given a reference unit and the required conversion factor.

Conversion of Models to a New Unit System

Function Introduced:

- **ProMdlPrincipalunitsystemSet()**

Use the function `ProMdlPrincipalunitsystemSet()` to change the principal system of units assigned to the solid model. The options available for the conversion are:

- `PRO_UNITCONVERT_SAME_DIMS`—Specifies the option to keep the dimension values despite the change in units.
- `PRO_UNITCONVERT_SAME_SIZE`—Specifies the option to scale the dimension values to keep the same size for the model.

Conversion of a system of units may result in regeneration failures due to the modification of dimensions, parameters, and relations. `ProMdlPrincipalunitsystemSet()` does not support a flag to undo the changes made by the unit system conversion, and will always bring the **Fix Model** interface to fix any regeneration failure. Therefore use this function only in interactive mode applications.While updating the principal system of units in an assembly environment, update the system of units in the following order:

1. Update the system of unit for all the parts separately. Update the parts using the following procedure:
   a. Retrieve the parts.

b. Update the units.
c. Save the part and erase it from the current session.

2. Update all the sub-assemblies, that either need to be changed, or contain already processed components.
3. Update the topmost level assembly.

Note

The initial units for an assembly are those of its base component. If, however, the units of the base component have been changed, the assembly units do not automatically change. You must also modify the units of the assembly. You cannot change the units of an assembly containing assembly features that intersect a part.

Mass Properties

Function Introduced:

- **ProSolidMassPropertyGet()**
- **ProSolidBodyMassPropertyGet()**
- **ProSolidBodyDensityGet()**
- **ProSolidMassPropertyWithDensityGet()**
- **ProAssemblySolidMassPropertyGet()**

In Creo Parametric 7.0.0.0 and later, the density parameter for any material is `PTC_MASS_DENSITY`. When you edit the density of a material, the value of this parameter is updated. The alternate mass property parameter for an assembly, part, or body is `PRO_MP_ALT_DENSITY`. The reported density parameter for an assembly, part, or body is `PRO_MP_DENSITY`. In the case of an assembly or a part with different materials, the value of this parameter is the average density.

The function `ProSolidMassPropertyGet()` provides information about the distribution of mass in the part or assembly.

The function `ProSolidBodyMassPropertyGet()` calculates the mass properties of a body in the specified coordinate system. The input parameter *body* is the handle to a part or an assembly.

Both the functions provide the mass distribution information relative to the specified coordinate system datum *csys_name*. If the value of the parameter *csys_name* is `NULL`, the default coordinate system is used.

The functions `ProSolidMassPropertyGet()` and `ProSolidBodyMassPropertyGet()` return the information in the structure `ProMassProperty`, declared in the header file `ProSolid.h`.

The `ProMassProperty` structure contains the following fields (all doubles):

- `volume`—The volume.
- `surface_area`—The surface area.
- `density`—The density is not defined until a material with well-defined density is assigned.
- `mass`—The mass.
- `center_of_gravity[3]`—The center of gravity (COG).
- `coor_sys_inertia[3][3]`—The inertia matrix.
- `coor_sys_inertia_tensor[3][3]`—The inertia tensor.
- `cg_inertia_tensor[3][3]`—The inertia about the COG.
- `principal_moments[3]`—The principal moments of inertia (the eigenvalues of the COG inertia).
- `principal_axes[3][3]`—The principal axes (the eigenvectors of the COG inertia).

The function `ProSolidBodyDensityGet()` determines the density of the material assigned to a body. The input parameters *body* is the handle to the body.

Note

If a material is already assigned to the part, the output of the function is the density of the material that is assigned to the body. The density is measured in the units of the model. The density of the body is always the density of the material assigned to the body.

The function `ProSolidMassPropertyWithDensityGet()` calculates the mass properties of a part or an assembly in the specified coordinate system. This function does not impact the mass properties of a solid. The input arguments are as follows:

- *solid*—Handle to the part or assembly specified by the `ProSolid` object.
- *csys_name*—Name of the coordinate system. If this is `Null`, the function uses the default coordinate system.
- *dens_use_flag*—Value of the density flag specified using the enumerated data type `ProMPDensUse` and the valid values are as follows:
  - `PRO_MP_DENS_DEFAULT`—Calculate the mass properties using the material density.

    - PRO_MP_DENS_USE_ALWAYS—Calculate the mass properties using the specified density, even if material has a defined density.
    - PRO_MP_DENS_USE_IF_MISSING—Calculate mass properties using specified density, even if material does not have a defined density.
- *density*—Density used while calculating mass properties depending on the value specified for the input argument dens_use_flag.

The function ProAssemblySolidMassPropertyGet() calculates the mass properties of a solid that is referenced by the specified coordinate system selection. The input arguments follow:

- *solid* —The handle to top assembly or component/sub-assembly.
- *csys_sel*—Selection of coordinate system specified using the array of ProSelection object. If this is NULL, the function uses the default coordinate system of the specified solid.

Solid Postfix Identifiers

Functions Introduced:

- **ProSolidToPostfixId()**
- **ProPostfixIdToSolid()**

The postfix identifier of a solid is the integer run-time identifier used in relations to make the names of its dimensions unique in the context of a parent assembly. Creo Parametric automatically updates these values when they are used in relations. The function ProSolidToPostfixId() gives you the identifier for the solid in session. The ProPostfixIdToSolid() function provides the solid handle, given the identifier.

Part Objects

The object ProPart is an instance of ProSolid. It is an opaque handle that can be cast to a ProSolid or ProMdl so you can use any of the functions for those objects.

Density

Functions Introduced:

- **ProPartDensityGet()**

Superseded Functions:

- **ProPartDensitySet()**

The density of a part is used in many calculations inside of Creo Parametric, including mass properties calculations and shrinkwrap export. The function `ProPartDensityGet()` returns the calculated or reported density that is defined by the parameter `PRO_MP_DENSITY`.

In Creo Parametric 7.0.0.0 and later, the function `ProPartDensitySet()` is deprecated. Use the functions `ProMaterialCurrentSet()` and `ProMaterialPropertySet()` instead.

For more information about materials, refer to Accessing Material Data on page 119.

Example 4: Writing the Mass of a Given Part to the Model Tree

The sample code in `UgSolidInfoMass.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_solid` demonstrates the use of `ProPartDensityGet()`, `ProSolidMassPropertyGet()` and several units related functions. It writes the mass of the given part to the model tree, along with the appropriate units for the mass value.

Material Objects

Creo Parametric TOOLKIT enables you to programmatically access the material properties of parts. Using the Creo Parametric TOOLKIT functions, you can do the following actions:

- Create or delete materials.
- Set the current material.
- Retrieve and set the material types and properties.
- Read and write to material files.

To enable access to materials, Creo Parametric TOOLKIT uses the following objects:

- `ProMaterial`—A structure that contains a material name and the part (`ProSolid` object) to which it is assigned. This handle is used in older material functions.
- `ProMaterialItem`—Another name for a `ProModelitem`, it contains the material owner, ID, and type (`PRO_RP_MATERIAL`).

To convert between `ProMaterial` and `ProMaterialItem`, use `ProModelitemByNameInit()` to obtain the item from the material owner and name. To obtain a `ProMaterial` from a `ProMaterialItem`, use `ProModelitemNameGet()` to get the material name.

Accessing Material Data

Functions Introduced:

- **ProMaterialCreate()**
- **ProPartMaterialsGet()**
- **ProMaterialDelete()**
- **ProMaterialCurrentGet()**
- **ProMaterialCurrentSet()**

The function `ProMaterialCreate()` creates a new material with the name you specify, and sets the default values within an associated `ProMaterialdata` object. Your application must set the correct material properties in the fields of the `ProMaterialdata` structure.

The input arguments of this function are as follows:

- *part* — Specifies the part.
- *matl_name* — Specifies the material name.
- *p_matl_data* — This argument has been deprecated. Pass `NULL` to create an empty material item whose properties can be set by `ProMaterialPropertySet()`.

The function `ProPartMaterialsGet()` obtains an array containing material names that exist in a part database. Note that you must use `ProArrayAlloc()` to allocate memory for this array. To remove a specified material from the part's database, call the function `ProMaterialDelete()`.

The current material of a part determines the material properties that will be used in some computational analyses of that part. Although multiple materials can be stored in a part database, only one material can be current. The function `ProMaterialCurrentGet()` gets the handle for the master material of the specified part. To set the master material, call the function `ProMaterialCurrentSet()`.

Note

- When the master material on a part with a single body is changed, the appearance, density, and sheetmetal properties of the body are updated. When you create a new body, it is automatically assigned the material assigned to the part. You can also explicitly assign a material to a body. In this case, even when the master material on the model is changed, the appearance, density, and sheetmetal properties of the body are not changed.
- By default, when assigning a material to a sheetmetal part, the function `ProMaterialCurrentSet()` modifies the values of the sheetmetal properties such as Y factor and bend table according to the material file definition. This triggers a regeneration and a modification of the developed length parameters of the sheetmetal part. To prevent this regeneration, set the value of the configuration option `material_update_smt_bend_table` to `never_replace`. To trigger a regeneration and a modification of the developed length parameters of the sheetmetal part, set the configuration option `material_update_smt_bend_table` to `always_replace`. The default value is `always_replace`.

  See the Creo Parametric Sheetmetal online help for more information on Bend Allowance.
- The function `ProMaterialCurrentSet()` may change the model display, if the new material has a default appearance assigned to it.
- The function may also change the family table, if the parameter `PTC_MASTER_MATERIAL` is a part of the family table.
- You can still use the legacy parameter `PTC_MASTER_MATERIAL`, however, these legacy parameters do not appear correctly in calculations and reports when you are working with a part that uses multiple materials.

Material Types and Properties

The enumerated type `ProMaterialPropertyType` contains the material types and material property types.

The material type is given by `PRO_MATPROP_TYPE` that takes the following values:

- PRO_MATERIAL_TYPE_STRUCTURAL_ISOTROPIC—Specifies a material with an infinite number of planes of material symmetry, making the structural material properties equal in all directions.
- PRO_MATERIAL_TYPE_STRUCTURAL_ORTHOTROPIC—Specifies a material with symmetry relative to three mutually perpendicular planes for structural material properties.
- PRO_MATERIAL_TYPE_STRUCTURAL_TRANS_ISOTROPIC—Specifies a material with rotational symmetry about an axis for structural material properties. These properties are equal for all directions in the plane of isotropy.
- PRO_MATERIAL_TYPE_THERMAL_ISOTROPIC—Specifies a material with an infinite number of planes of material symmetry, making the thermal material properties equal in all directions.
- PRO_MATERIAL_TYPE_THERMAL_ORTHOTROPIC—Specifies a material with symmetry relative to three mutually perpendicular planes for thermal material properties.
- PRO_MATERIAL_TYPE_THERMAL_TRANS_ISOTROPIC — Specifies a material with rotational symmetry about an axis for thermal material properties. These properties are equal for all directions in the plane of isotropy.
- PRO_MATERIAL_TYPE_FLUID—Specifies a material with fluid properties.

The material subtype is given by PRO_MATPROP_SUB_TYPE that takes the following values:

- PRO_MATERIAL_SUB_TYPE_LINEAR—Specifies the linear elastic material type. This is the default value.
- PRO_MATERIAL_SUB_TYPE_HYPERELASTICSpecifies the hyperelastic (non-linear) material types, such as rubber, that exhibit instantaneous elastic response to large strains.
- PRO_MATERIAL_SUB_TYPE_ELASTOPLASTIC—Specifies the elastoplastic (non-linear) material types, such as metals, with the following isotropic hardening laws:
    - Perfect Plasticity—Given by the value PRO_MATERIAL_HARDENING_PERFECT_PLASTICITY
    - Linear Hardening—Given by the value PRO_MATERIAL_HARDENING_LINEAR_HARDENING
    - Power Law—Given by the value PRO_MATERIAL_HARDENING_POWER_LAW
    - Exponential Law—Given by the value PRO_MATERIAL_HARDENING_EXPONENTIAL_LAW

The above three subtypes are available only for PRO_MATERIAL_TYPE_STRUCTURAL_ISOTROPIC and PRO_MATERIAL_TYPE_THERMAL_ISOTROPIC material types.

From Creo Parametric 3.0 onward, two additional material types PRO_MATERIAL_FATIGUE_MAT_TYPE_FERROUS and PRO_MATERIAL_FATIGUE_MAT_TYPE_OTHER have been added for fatigue analysis. The material types PRO_MATERIAL_FATIGUE_MAT_TYPE_UNALLOYED_STEELS and PRO_MATERIAL_FATIGUE_MAT_TYPE_LOW_ALLOY_STEELS are obsolete. Use the material type PRO_MATERIAL_FATIGUE_MAT_TYPE_FERROUS instead. The following surface finish types for fatigue analysis are also obsolete:

- PRO_MATERIAL_FATIGUE_FINISH_FORGED
- PRO_MATERIAL_FATIGUE_FINISH_WATER_CORRODED
- PRO_MATERIAL_FATIGUE_FINISH_SEA_WATER_CORRODED
- PRO_MATERIAL_FATIGUE_FINISH_NITIRIDED
- PRO_MATERIAL_FATIGUE_FINISH_SHOT_PEENED

PTC recommends that you review your existing Creo Parametric TOOLKIT applications and modify the code as appropriate to ensure that the applications work correctly for the fatigue materials and material finish types.

Functions Introduced:

- **ProMaterialPropertyGet()**
- **ProSolidBodyMaterialGet()**
- **ProSolidBodyMaterialSet()**
- **ProUnitExpressionGet()**
- **ProMaterialPropertySet()**
- **ProUnitInitByExpression()**
- **ProMaterialDescriptionGet()**
- **ProMaterialDescriptionSet()**
- **ProMaterialPropertyDelete()**

The functions ProMaterialDataGet() and ProMaterialDataSet() have been deprecated, and do not support all of the available material properties. PTC recommends that for accessing material properties, you convert the ProMaterial type to a model item using ProModelitemByNameInit(), and use ProMaterialPropertyGet() and ProMaterialPropertySet() the properties of that item, respectively.

A part created in Creo Parametric 7.0.0.0 and later, can contain multiple solid bodies where each body can have its own material assignment. A part created in an earlier release of Creo Parametric contains one body and one material is assigned to the part.

The function `ProSolidBodyMaterialSet()` assigns a material to the specified body. You can set the default material by specifying the value of the system parameter `PTC_MASTER_MATERIAL` as `PTC_SYSTEM_MTRL_PROPS`. In legacy parts, the value of `PTC_MASTER_MATERIAL` is the material assigned to the part. The input arguments follow:

- `body`—Body for which the material needs to be assigned.
- `mtl`—Name of the material that needs to be assigned to the body.

Use the function `ProSolidBodyMaterialGet()` to retrieve the information of the material assigned to the body.

Note

Refer to the Creo Parametric online help for more information about Materials.

If you do not assign a material to a body, Creo Parametric assigns `PTC_GENERIC_MATERIAL` material to the body. The density of this material as well as other properties are empty.

The function `ProMaterialPropertyGet()` returns the value and the units for a material property.

Note

The name of the units returned can be the name of a Creo Parametric unit, which may not be obviously understood by a user. Use `ProUnitExpressionGet()` to change this name to familiar units.

Use the function `ProMaterialPropertySet()` to create or modify a material property. It has the following input parameters:

- *p_material*—Specifies the material as defined by `ProMaterialItem`.
- *prop_type*—Specifies the material property type as defined by `ProMaterialPropertyType`.
- *p_value*—Specifies the material property value.
- *p_units*—Specifies the material property units.

Note

This function expects the Creo Parametric unit name for some unit properties. To obtain this name, pass the user-visible units through `ProUnitByExpressionInit()`.

The following table displays the possible combinations of arguments for *p_value* and *p_units*:

p_value	p_units	Is the property already created in the material?	Result
Any value	Appropriate units for this property, or `NULL`, if the property is unitless	`NO`	Property is created with the given units and value.
Any value	`NULL`	`NO`	Property is created with the given value using the appropriate units from the owner model.
Any value	Current units for this property, or `NULL`, if the property is unitless	`YES`	Property value is changed to the new value.
Any value	`NULL`	`YES`	Property value is changed to the new value (which is interpreted as being in the units from the owner model)
The current value	New appropriate units	`YES`	Property units are changed but the value is interpreted as being for the new units.
`NULL`	New appropriate units	`YES`	Property units are changed and the current value is converted to the new units.

Note

When using `ProMaterialPropertySet()` to change the sheetmetal Y-factor or bend table assigned to the current material, pass the current material to `ProMaterialCurrentSet()` again to force Creo Parametric to update the length calculations developed by sheetmetal.

Use the function `ProMaterialDescriptionGet()` to get the material description. This property is also accessible as the material property `PRO_MATPROP_MATERIAL_DESCRIPTION`.

Use the function `ProMaterialDescriptionSet()` to set the material description.

Use the function `ProMaterialPropertyDelete()` to remove a property from the material definition.

Material Input and Output

Functions Introduced:

- **ProMaterialfileWrite()**
- **ProMaterialfileRead()**

Material properties are frequently stored in text files accessible for repeated assignment to parts. Creo Parametric TOOLKIT includes functions that write to and read these files.

The function `ProMaterialfileWrite()` writes the information contained in a `ProMaterial` object to a file with the specified name.

The format of this file is the new material file format which is consistent with the Creo Parametric materials library.

The function `ProMaterialfileRead()` reads from a material file, the properties of the material with the specified name. The name of the file read can be either:

- `<name>.mtl`—Specifies a new material file format.
- `<name>.mat`—Specifies an old material file format (pre-Wildfire 3.0).

If the material is not already in the part database, `ProMaterialfileRead()` adds the material to the database after reading the material file. If the material is already in the database, the function replaces the material properties in the database with those contained in the material file.

Example 5: Working with Materials and Material Properties

The sample code in `UgMaterial.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_param` shows how to work with materials and material properties.

Example 6: Creating a Non-linear Material

The sample code in `UgSolidMaterial.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_solid` shows how to create a non-linear material and assign it to a solid.

4

Core: Solid Body

This chapter describes how to access, create, or delete bodies in a model.

Introduction to Solid Body

In Creo Parametric 7.0.0.0 and later, the term “solid body” denotes a container object for solid geometry.

In earlier versions of Creo Parametricall solid geometry in a part is considered as one piece of a single material, even when the geometry has disjoint volumes. Starting in Creo Parametric 7.0.0.0, you can create parts that contain one or more geometric bodies. Each body can be handled individually, and can have different characteristics. For example, you can assign a different material to each body.

Bodies contain only solid geometry. Nonsolid entities, like datums, curves, and quilts, are not contained in any body.

When you create a new part, it has an empty body in it. This body will contain the solid geometry created by the features. If no solid geometry is created, or as long as the part contains only nonsolid geometry, this body remains empty. A part must always have at least one body in it.

When you retrieve a part that was created using a version earlier of Creo Parametric , it shows a single body in it. This body contains all the solid geometry in the part, if any exists.

Using Creo Parametric solid body feature, you can create parts with one or more geometric bodies. A body consists of solid geometry of the same material. You can assign a different material to each body and a single part can have more than one material.

Solid Body Objects

The structure `ProSolidBody` describes the contents the solid body object. This object uses the same declaration as the `ProModelitem` object, which is as follows:

```
typedef struct pro_model_item
{
  ProType  type;
  int      id;
  ProMdl   owner;
} ProSolidBody;
```

States of bodies

When a body has geometry, it does not have a special state. A body can have a state derived from features and geometry.

- `No Contributing Features`— No feature contributes geometry to the body.
- `No Geometry`— Features which contribute to the body, however other features cut the entire geometry of this body. Or all the contributing features are suppressed.
- `Construction`—Does not participate in mass properties calculations and is not considered in global or volume interference analysis or in collision detection. The construction state does not change the other properties of the body. Setting a body to construction state is reversible.

Creating a Body

A new part is created with a default body. This body is in the `No Contributing Features` state. When you add solid geometry to this part, you can add to this body, or create a new body. You can delete bodies that are in the `No Contributing Features` state. When you delete a body, it is removed from the part and the **Bodies** folder in the model tree. The parameters and relations for the body are deleted.

Functions introduced:

- **ProSolidBodyCreate()**
- **ProSolidBodiesCollect()**
- **ProSolidDefaultBodyGet()**
- **ProSolidDefaultBodySet()**
- **ProSolidBodySurfaceVisit()**
- **ProSolidBodyDelete()**
- **ProSolidBodyStateGet()**
- **ProSolidBodyIsConstruction()**
- **ProSolidBodyConstructionSet()**
- **ProSolidBodyOutlineGet()**
- **ProSolidBodyIsSheetmetal()**

The function `ProSolidBodyCreate()` creates a new body. The input argument *sld* is the solid owner on which the body needs to be created. The output argument `body` is the body that is created.

Refer to the Creo Parametric online help for more information about body creation.

The function ProSolidBodiesCollect() collects all the bodies in the specified solid.

The function ProSolidDefaultBodyGet() returns the default body in the specified solid.

The function ProSolidDefaultBodySet() sets the specified body as default body in the specified solid. The input argument default_body is the body to be set as the default body.

Use the function ProSolidBodySurfaceVisit() to visit the surfaces that are included in the specified body.

The function ProSolidBodyDelete() deletes the body in the specified solid. When you delete a body, it is removed from the part and the **Bodies** folder in the model tree.

The function ProSolidBodyStateGet() returns the state of the body and is defined by the enumerated data type ProSolidBodyState and the valid values are:

- PRO_BODY_STATE_MISSING
- PRO_BODY_STATE_CONSUMED
- PRO_BODY_STATE_NO_CONTR_FEAT
- PRO_BODY_STATE_NO_GEOMETRY
- PRO_BODY_STATE_ACTIVE

Use the function ProSolidBodyIsConstruction() to check if the specified body is a construction body.

Use the function ProSolidBodyConstructionSet() to set the specified body as a construction body. The function returns the error PRO_TK_NO_CHANGE if the body is already a construction body.

Use the function ProSolidBodyOutlineGet() to retrieve the regeneration outline of a solid body, with respect to the base coordinate system orientation. This outline defines the boundary box of the body. The function returns PRO_TK_E_NOT_FOUND if the solid body is empty.

The function ProSolidBodyIsSheetmetal() checks if the specified body is an active sheetmetal body.

In Creo Parametric 7.0, a sheetmetal part can have a single sheetmetal body and any number of solid bodies.

Listing Features

Functions Introduced:

- **ProSolidBodyFeaturesGet()**

The function `ProSolidBodyFeaturesGet()` lists all the features that are associated with the specified body. The output argument `features` is a `ProArray` of features.

The function returns the error `PRO_TK_E_NOT_FOUND` if there are no contributing features associated with the body.

Multibody Operations

Each body has its own geometry. You can perform geometric operations such as splitting a body or merging with other bodies. Bodies contribute to the mass properties of the model. You can select bodies as references for features. You can split a body into two bodies and also perform move or copy operations on bodies. The geometry of the original body is divided between the original body and the new body. For more information on body options and body operations, refer to the Element Trees: Solid Body on page 1055 chapter.

You can use the Boolean Operations feature to perform geometric operations such as:

- **Merge** —Combines the geometry of two or more bodies into one body.
- **Intersect**—Keeps the geometry that is shared by two or more bodies.
- **Subtract**—Removes the geometry of one body from one or more bodies.

5

Core: Features

This chapter describes the Creo Parametric TOOLKIT functions that deal with features as a whole and the way they relate to each other.

Access to the geometry objects created by features is described in the Core: 3D Geometry on page 170 chapter.

Access to the internal structure of a feature is described in the Element Trees: Principles of Feature Creation on page 764 chapter.

Feature Objects

Function Introduced:

- **ProFeatureInit()**

Features are represented by the object `ProFeature`, which is declared as a DHandle, or data handle. It shares the same declaration as `ProModelitem` and `ProGeomitem`, and therefore contains the type and integer identifier as fields in the structure.

Like `ProGeomitem`, `ProFeature` is an instance of `ProModelitem`. `ProFeature` objects are contained in `ProSolid` objects, and contain `ProGeomitem` objects.

You can create a new `ProFeature` handle using the function `ProFeatureInit()`.

Visiting Features

Function Introduced:

- **ProSolidFeatVisit()**

The function `ProSolidFeatVisit()` enables you to visit all the features in a part or assembly. It visits not only those features visible to the user, but also features used internally for construction purposes. To skip over such internal, “invisible” functions, call `ProFeatureVisibilityGet()`.

Note that the function `ProSolidFeatstatusGet()` (described in detail in the Core: Solids, Parts, and Materials on page 92 chapter.) provides an array of integer identifiers for all the features in a solid, thereby offering an alternate way of finding all the features.

Feature Inquiry

Functions Introduced:

- **ProFeatureTypeGet()**
- **ProFeatureSubtypeGet()**
- **ProFeatureTypenameGet()**
- **ProFeatureStatusGet()**
- **ProFeatureStatusflagsGet()**
- **ProFeatureIsIncomplete()**
- **ProFeatureIsNcseq()**
- **ProFeatureSolidGet()**

- **ProFeatureChildrenGet()**
- **ProFeatureParentsGet()**
- **ProFeatureSelectionGet()**
- **ProFeatureHasGeomchks()**
- **ProFeatureIsReadonly()**
- **ProFeatureIsEmbedded()**
- **ProInsertModeIsActive()**
- **ProFeatureCopyinfoGet()**
- **ProFeatureZoneGet()**
- **ProFeatureZonesectionCreate()**
- **ProFeatureZonesectionGet()**
- **ProZoneReferenceFree()**
- **ProZoneReferenceArrayFree()**
- **ProFeatureZonesectionWithflipCreate()**
- **ProFeatureZoneXsecgeomGet()**
- **ProFeatureZoneXsecGeomArrayFree()**
- **ProModelitemIsZone()**
- **ProFeatureIsInFooter()**
- **ProFeatureToFooterMove()**
- **ProFeatureFromFooterMove()**
- **ProFeatureIsComponentLike()**

As described earlier, the function `ProSolidFeatVisit()` finds all the features belonging to a part or an assembly. The feature inquiry functions provide more information about a particular feature.

The function `ProFeatureTypeGet()` provides the type of the feature. This feature type uses the data type `ProFeattype`, which is really an integer that takes defined values such as the following:

- `PRO_FEAT_FIRST_FEAT`
- `PRO_FEAT_HOLE`
- `PRO_FEAT_SHAFT`
- `PRO_FEAT_ROUND`

See the include file `ProFeatType.h` for the list of defined values.

The function `ProFeatureTypenameGet()` returns the name of the feature type. Given a `ProFeature` pointer to a specific feature, this function returns the name of the feature type, for example, `CHAMFER`, `DATUM`, `COORDINATE SYSTEM`, and so on. Arguments to this function must not be `NULL`.

The function `ProFeatureSubtypeGet()` provides the subtype (such as sheet metal) of a specified feature. Note that not all features support subtypes. This is like viewing valid model subtypes by opening the **Model Tree** settings command in Creo Parametric. Click **Settings ► Tree Columns** menu and then select **Feat Subtype** in the **Model Tree Columns** dialog box, as an additional display column.

The function `ProFeatureStatusGet()` classifies the feature according to the following status values:

- `PRO_FEAT_ACTIVE`—An ordinary feature.
- `PRO_FEAT_SUPPRESSED`—A suppressed feature.
- `PRO_FEAT_FAMTAB_SUPPRESSED`—A feature suppressed due to the family table settings.
- `PRO_FEAT_SIMP_REP_SUPPRESSED`—A feature suppressed due to the simplified representation.
- `PRO_FEAT_PROG_SUPPRESSED`—A feature suppressed due to Pro/PROGRAM.
- `PRO_FEAT_INACTIVE`—A feature that is not suppressed, but is not currently in use for reasons other than the ones identified above.
- `PRO_FEAT_UNREGENERATED`—A feature that has not yet been regenerated. This is due to a regeneration failure or if the status is obtained during the regeneration process.
- `PRO_FEAT_INVALID`—The feature status could not be retrieved.

The function `ProFeatureStatusflagsGet()` retrieves the bitmask containing one or more of the following feature status bit flags for a specified feature:

- `PRO_FEAT_STAT_INVALID`—Specifies an invalid feature.
- `PRO_FEAT_STAT_INACTIVE`—Specifies an inactive feature. If the bit flag is set to 0, then it means an active feature.
- `PRO_FEAT_STAT_ACTIVE`—Specifies an active feature.
- `PRO_FEAT_STAT_FAMTAB_SUPPRESSED`—Specifies a feature suppressed due to the family table settings.
- `PRO_FEAT_STAT_SIMP_REP_SUPPRESSED`—Specifies a feature suppressed due to the simplified representation.
- `PRO_FEAT_STAT_PROG_SUPPRESSED`—Specifies a feature suppressed due to Pro/PROGRAM.

- PRO_FEAT_STAT_SUPPRESSED—Specifies a suppressed feature.
- PRO_FEAT_STAT_UNREGENERATED—Specifies an active feature that has not yet been regenerated. This is due to a regeneration failure or if the status is obtained during the regeneration process.
- PRO_FEAT_STAT_FAILED—Specifies a failed feature.
- PRO_FEAT_STAT_CHILD_OF_FAILED—Specifies a child of a failed feature.
- PRO_FEAT_STAT_CHILD_OF_EXT_FAILED—Specifies a child of an external failed feature.

The function ProFeatureIsIncomplete() tells you whether a specified feature is incomplete. An incomplete feature is one that has been created by using ProFeatureCreate() from a Creo Parametric TOOLKIT application, but which does not yet contain all the necessary feature elements to allow regeneration.

The function ProFeatureIsNcseq() determines whether a feature is a Creo NC sequence.

The ProFeatureSolidGet() function provides the identifier of the solid that owns the specified feature.

The ProFeatureChildrenGet() and ProFeatureParentsGet() functions get the children and parents of the specified feature. For these functions, the parent of a feature means a feature it directly depends on, and a child is a feature that directly depends on it. This differs from the Creo Parametric command **Info ► Feature**, which also shows indirect dependencies.

The function ProFeatureSelectionGet() is used for features that were created in a part as a result of a feature in a parent assembly. For example, if you create a hole in Assembly mode, then select a part to be intersected by that hole, the geometry of the hole is visible to Creo Parametric TOOLKIT as belonging to the part, even if the original feature is specified as being visible at the assembly level. This geometry—a list of the surfaces forming the hole—belongs to a feature in the part whose type is PRO_FEAT_ASSEM_CUT. The function ProFeatureSelectionGet(), when applied to that part feature, identifies the assembly, and the path down through it to the part in question, which contains the original feature.

During regeneration, Creo Parametric performs geometry checking to prevent regeneration errors. The geometry check process identifies features that could cause problems if the part or assembly is modified, but which do not cause regeneration failure in the model in its present state. The ProFeatureHasGeomchks() function outputs a variable of type ProBoolean that indicates whether a particular feature, identified as an input argument to the function, has geometry checks.

The function `ProFeatureIsReadonly()` provides information about the read status of the specified feature. Its first argument is a pointer to the feature's (`ProFeature`) handle. If the feature is read only, the function outputs a `ProBoolean` with the value `PRO_B_TRUE`; otherwise, the value is `PRO_B_FALSE`.

The function `ProFeatureIsEmbedded()` identifies whether the feature is an embedded datum. Embedded features are visible in the model tree, but cannot be used as reference parents for features other than the feature into which they are embedded.

To determine whether insert mode is active in a specified solid, use the function `ProInsertModeIsActive()`. If activated, features are inserted into the feature list after the feature specified when `ProFeatureInsertModeActivate()` was called. New features continue to be inserted until you call the function `ProInsertModeCancel()`. See the section Manipulating Features on page 138 for more information about insert mode.

The function `ProFeatureCopyinfoGet()` returns information about a copied feature. The information includes the type of copy operation, dependency, source feature, and additional features copied in the same operation. This function supersedes the `Pro_copy_info` structure returned by the Pro/Develop function `prodb_feature_info()`.

The function `ProFeatureZoneGet()` returns the following parameters related to a feature zone:

- *p_planes*—`ProArray` of planes.
- *p_oper_arr*—`ProArray` of operations; where `0` specifies intersection of half spaces and `1` specifies union of half spaces. Creo Parametric retains the material that belongs to the intersection or union of the half spaces of the planes.

The function `ProFeatureZonesectionCreate()` creates a zone feature handle using reference planes and operations. The input arguments are as follows:

- *p_solid*—A handle to the model.
- *zone_refs*—An array of zone reference planes of type `ProZoneReference`. The structure `ProZoneReference` contains the geometric ID of the reference zone plane, the value for the operation, that is, `0` or `1` and the member ID of the part to which the reference plane belongs to. Pass `NULL` to `memb_id_tab` if the feature is owned by the part on which the zone is being created.

Creo Parametric retains the material that belongs to the intersection or union of the half spaces of the reference planes.

- *zone_name*—The name of the zone feature handle. If a zone with the specified name exists, then the function returns the error `PRO_TK_E_FOUND` and the zone is not created.

The function `ProFeatureZonesectionGet()` returns the zone references for the specified feature. The output argument `p_zone_refs` contains an array of planes of type `ProZoneReferenceWithflip`. The structure `ProZoneReferenceWithflip` contains:

- The geometric ID of the reference zone plane.
- The value for the operation, where `0` specifies intersection of half spaces that is, the AND operator and `1` specifies union of half spaces that is, the OR operator.
- The member ID of the part to which the reference plane belongs.
- The side of the plane where the model is kept. `1` indicates positive normal of the plane and `-1` indicates the opposite side.

Use the function `ProZoneReferenceFree()` to free the memory allocated to the zone reference data.

Use the function `ProZoneReferenceArrayFree()` to free the `ProArray` of zone reference data.

The function `ProFeatureZonesectionWithflipCreate()` creates a zone feature using reference planes and operations. This function allows you to flip the direction of zone planes while creating the zone feature.

The function `ProFeatureZoneXsecgeomGet()` creates an array of cross section geometry of type `ProXsecGeometry` for each zone plane. It returns an array of these arrays in the specified zone feature. Use the function `ProFeatureZoneXsecGeomArrayFree()` to free the memory allocated for the `ProArray` of `ProArrays` of type `ProXsecGeometry`.

The function `ProModelitemIsZone()` checks if the specified model item is a zone feature. Specify the handle to the model item as the input argument of this function.

Use the function `ProFeatureIsInFooter()` to check if the specified feature is currently located in the model tree footer. The footer is a section of the model tree that lists certain types of features such as, component interfaces, annotation features, zones, reference features, publish geometry, and analysis feature. The features in the footer are always regenerated at the end of the feature list. You can move features, such as, reference features, annotation features, and so on, to the footer. Some features, such as, component interfaces, zones, and so on, are automatically placed in the footer. Refer to the Creo Parametric online Help for more information on footer. Refer to the Creo Parametric online Help for more information on footer.

Use the function `ProFeatureToFooterMove()` to move the specified feature into the model tree footer.

Use the function `ProFeatureFromFooterMove()` to move the specified feature out of the model tree footer.

Some features behave like components because they have some properties that are similar to those of components. These features have some association with a solid model and are interpreted as placed components. When a component is placed it means it has been explicitly positioned at some location in the assembly. Examples of such features are solid welds, physical sensors, and so on. Solid welds organize their geometry as a special internal solid model, which gives them component-like characteristics. Similarly, physical sensors represent actual hardware that is placed on the model to measure parameters. Use the function `ProFeatureIsComponentLike()` to identify components and other features that behave like components. Refer to the chapter Assembly: Basic Assembly Access on page 1130 for more information on placed components.

Feature Geometry

Functions Introduced:

- **ProFeatureGeomitemVisit()**
- **ProGeomitemFeatureGet()**

For information about feature geometry, see the chapter Core: 3D Geometry on page 170.

Manipulating Features

Functions Introduced:

- **ProFeatureDelete()**
- **ProFeatureSuppress()**
- **ProFeatureResume()**
- **ProFeatureRedefine()**
- **ProFeatureInsertModeActivate()**
- **ProInsertModeCancel()**
- **ProFeatureReadonlySet()**
- **ProFeatureReadonlyUnset()**
- **ProFeatureReorder()**
- **ProFeatureNumberGet()**

The functions `ProFeatureDelete()` and `ProFeatureSuppress()` act like the right-mouse button Creo Parametric commands **Delete** and **Suppress**, except they do not repaint the window. You can process many features in a single call using an input of type `ProFeatlist`. Each of these functions takes an array of options as the input that indicates whether to also delete or suppress features dependent on those being acted on directly. The options used while deleting or suppressing features are as follows:

- `PRO_FEAT_DELETE_NO_OPTS`—Delete or suppress the features without deleting or suppressing their dependent children features. This may result in regeneration failures. Use the option `PRO_FEAT_DELETE_FIX`, or one of the CLIP options to fix these failures.
- `PRO_FEAT_DELETE_CLIP`—Delete or suppress the features along with their dependent children features.
- `PRO_FEAT_DELETE_FIX`—Delete or suppress the features without deleting or suppressing their dependent children features. The fix model user interface will be prompted in case of a regeneration failure. This option must be used only in the Resolve mode. Otherwise, the function returns `PRO_TK_BAD_CONTEXT`.
- `PRO_FEAT_DELETE_RELATION_DELETE`—Delete relations with obsolete dimensions.
- `PRO_FEAT_DELETE_RELATION_COMMENT`—Change relations with obsolete dimensions into comments.
- `PRO_FEAT_DELETE_CLIP_ALL`—Delete or suppress the features along with all the following features.
- `PRO_FEAT_DELETE_INDIV_GP_MEMBERS`—Individually delete or suppress the features out of the groups to which they belong. If this option is not included, the entire group of features is deleted or suppressed. This option can be included only if the option `PRO_FEAT_DELETE_CLIP` is also included.
- `PRO_FEAT_DELETE_CLIP_INDIV_GP_MEMBERS`—Individually delete or suppress the children of features out of the group to which they belong. If this option is not included, the entire group containing the features and their children is deleted or suppressed. This option can be included only if the options `PRO_FEAT_DELETE_CLIP` and `PRO_FEAT_DELETE_INDIV_GP_MEMBERS` are also included.
- `PRO_FEAT_DELETE_KEEP_EMBED_DATUMS`—Retain the embedded datums stored in a feature while deleting the feature using `ProFeatureDelete()`. If this option is not included, the embedded datums will be deleted along with the parent feature.

The function `ProFeatureRedefine()` is equivalent to the Creo Parametric command **Feature**>**Redefine**. Additionally, it can redefine an existing feature with the new element tree. The data passed in through the new element tree replaces the existing data in the feature.

Creo Parametric TOOLKIT provides access to the Creo Parametric feature insert mode functionality with the `ProFeatureInsertModeActivate()` and `ProInsertModeCancel()` functions. The function `ProFeatureInsertModeActivate()` takes a single argument—the handle to the feature after which new features are to be inserted. This feature becomes the last feature in the feature regeneration list. All features that had appeared after that feature are temporarily suppressed. New features are added after the (new) last feature. Feature insertion continues until insert mode is terminated with a call to `ProInsertModeCancel()`. Its first argument is a handle to the solid, and the second is a `ProBoolean` that enables you to specify whether suppressed features are to be resumed.

The function `ProFeatureReadonlySet()` assigns a read-only status to model features. Its only argument is a `ProFeature` handle that specifies the last feature in the feature list to be designated as read only. All preceding features are read only; all features following this feature have standard access. From Creo Parametric 3.0 onward, the features that are made read-only appear under a separate container node at the top of the model tree. The node has its label as **Read Only Features** and also has a padlock glyph associated with it.

The function `ProFeatureReadonlyUnset()` removes the read-only status from all features in the specified solid. From Creo Parametric 3.0 onward, the container node **Read Only Features** is dismissed from the model tree when the read-only status is removed.

The function `ProFeatureReorder()` enables you to change the position of one or more features in the feature regeneration sequence. Its input arguments are as follows:

- `ProSolid` *solid*—The handle to the solid owner of the features.
- `int` **feat_ids*—An array of feature identifiers that specifies the features to be reordered. The array should contain features that formed a contiguous sublist within the original feature regeneration list. If reordering a group, all the features in the group including the Group Header feature must be included in this array.
- `int`*n_feats*—The number of features to reorder.
- `int`*new_feat_num*—An integer that indicates the intended location of the first feature in the specified set after reorder. This integer is not the feature identifier, but rather the regeneration sequence number of the feature. You obtain this number by calling the function `ProFeatureNumberGet()`.

Use the function `ProSolidFeatstatusGet()` to get the current sequence and statuses. You must use care when you change the sequence of features. Unless you have advance knowledge of the relationship between the features you are reordering, you should use the functions `ProFeatureParentsGet()` and `ProFeatureChildrenGet()` before changing the feature order to ensure that no feature is reordered to be before its parent features.

Manipulating Features based on Regeneration Flags

Functions Introduced:

- **ProFeatureWithoptionsCreate()**
- **ProFeatureWithoptionsDelete()**
- **ProFeatureWithoptionsSuppress()**
- **ProFeatureWithoptionsResume()**
- **ProFeatureWithoptionsRedefine()**
- **ProFeatureWithoptionsReorder()**
- **ProFeatureInsertmodeWithoptionsActivate()**
- **ProInsertmodeWithoptionsCancel()**

The functions in this section enable you to create, delete, or manipulate the specified features in a solid, based on the bitmask specified by the input argument *flags*. The bitmask must contain one or more regeneration control bit flags of the type `PRO_REGEN_*` defined in `ProSolid.h`. Refer to the Regenerating a Solid on page 96 section in the Core: Solids, Parts, and Materials on page 92 chapter for more information on the bit flags.

From Pro/ENGINEER Wildfire 5.0 onwards, the functions listed above supersede the following functions described in the Manipulating Features on page 138 section.

- `ProFeatureCreate()`
- `ProFeatureDelete()`
- `ProFeatureSuppress()`
- `ProFeatureResume()`
- `ProFeatureRedefine()`
- `ProFeatureReorder()`
- `ProFeatureInsertModeActivate()`
- `ProInsertModeCancel()`

Use the superseding functions with the input argument *flags* set to `PRO_REGEN_NO_FLAGS` for the behavior that is similar to the one provided by the above deprecated functions.

Feature References

Functions Introduced:

- **ProFeatureReferenceEdit()**
- **ProMdlFeatBackupOwnerNamesGet()**
- **ProMdlFeatBackupRefMdlNamesGet()**
- **ProFeatureReferenceEditRefsGet()**
- **ProFeatureMdltreeDisplaynameGet()**

The function `ProFeatureReferenceEdit()` replaces the old references of a feature with new references based on the bitmask specified for its input argument *flags*. The bitmask must contain one or more regeneration control bit flags of the type `PRO_REGEN_*` defined in `ProSolid.h`. Refer to the Regenerating a Solid on page 96 section in the Core: Solids, Parts, and Materials on page 92 chapter for more information on the bit flags.

The function `ProMdlFeatBackupOwnerNamesGet()` returns the names of the models, along the model path, from the top model to the owner model for the specified feature. The input arguments are:

- *model*—Specifies the model, which contains the specified feature.
- *feature*—Specifies a feature whose references are to be retrieved.

The function `ProMdlFeatBackupRefMdlNamesGet()` returns the names of the models, along the model path, from the top model to the external reference model for the specified feature. Feature references can be from a local or external model. The system creates a geometry backup of the local and external references, which is used for information and display purposes. This function retrieves the model names from the backup information. The input arguments are:

- *model*—Specifies the model, which contains the specified feature.
- *feature*—Specifies a feature whose references are to be retrieved.
- *path*—Specifies the path as a `ProArray` of IDs of a subassembly or component from the top model to the reference model. Specify `NULL` for local reference.

  To give an example on how to specify the path, consider an assembly A, which has a component C1 with ID 9 and a subassembly S with ID 7. The

subassembly S has a component C2 with ID 11. If a feature under C1 references an object in the model of C2, the reference ID path must contain two IDs 7 and 11.

- *ref_id*—Specifies the ID of the external reference, which is referenced in the specified feature.

Refer to the Creo Parametric Assembly Design online help for more information on references and backup data.

The function `ProFeatureReferenceEditRefsGet()` returns an array of the original references of a feature that are used to perform the edit reference operation. The input arguments follow:

- *solid*—The part or assembly to which the features belong.
- *p_feat_handle*—The feature handle.
- *flags*—Indicates the type of references to collect. To collect all types of references, set the value to `PRO_EDITREF_REF_TYPE_ALL`.

The function returns the output argument `r_orig_ref_arr` as a `ProArray` of all the original references.

Use the function `ProReferencearrayFree()` to free the memory.

Use the function `ProFeatureMdltreeDisplaynameGet()` to retrieve the name of the node displayed in the model tree.

Feature Dimensions

Function Introduced:

- **ProFeatureDimensionVisit()**

The function `ProFeatureDimensionVisit()` visits all the dimensions which belong to the feature.

Note

Some feature dimensions are dependent on dimensions of other features. For example, sketch-based features with shared, patterned or external sections, dependent copied or mirrored features, and so on. In case of such dependent dimensions, use the function `ProDimensionParentGet()` to get the value of the parent dimension .

For more information about dimensions, refer to section Dimensions on page 566 from the chapter Annotations: Annotation Features and Annotations on page 541.

Manipulating Patterns

From the Pro/ENGINEER Wildfire release, the following changes are implemented in patterns.

Patterns as Features

Patterns are treated as features in Creo Parametric. Patterns are assigned a header feature of the type `PRO_E_PATTERN_HEAD`.

The Pattern feature in Pro/ENGINEER Wildfire effects the previous releases of Pro/ENGINEER as follows:

- Models containing patterns automatically get one extra feature of type `PRO_FEAT_PATTERN_HEAD` in the regeneration list. This changes the feature numbers of all the subsequent features, including those in the pattern.
- The pattern access functions such as `ProFeaturePatternGet()`, `ProPatternMembersGet()` and `ProPatternLeaderGet` are unaffected by the addition of the pattern header feature. The pattern leader is still the first geometric feature contained in the pattern.

The new function `ProPatternHeaderGet()` returns the header feature.

Fill Patterns

Creo Parametric uses the Fill type of pattern.

Functions Introduced:

- **ProFeaturePatternStatusGet()**
- **ProFeatureGrppatternStatusGet()**
- **ProFeaturePatternleaderGet()**
- **ProFeaturePatternGet()**
- **ProPatternDelete()**
- **ProPatternLeaderGet()**
- **ProPatternHeaderGet()**

The function `ProFeaturePatternStatusGet()` classifies the feature according to its possible role in a feature pattern. The possible values are as follows:

- `PRO_PATTERN_NONE`—The feature is not in a pattern.
- `PRO_PATTERN_LEADER`—The feature is the leader of a pattern.
- `PRO_PATTERN_MEMBER`—The feature is a member of a pattern.
- `PRO_PATTERN_HEADER`—The feature is the header of a pattern.

The function ProFeatureGrppatternStatusGet() does the equivalent for group patterns. The possible values are as follows:

- PRO_GRP_PATTERN_NONE—The feature is not in a group pattern.
- PRO_GRP_PATTERN_LEADER—The feature is the leader of a group pattern.
- PRO_GRP_PATTERN_MEMBER—The feature is a member of a group pattern.
- PRO_GRP_PATTERN_HEADER—The feature is the header of a group pattern.

The function ProFeaturePatternleaderGet() returns the pattern leader feature for the specified pattern member feature.

The function ProFeaturePatternGet() obtains the ProPattern handle for the pattern containing the specified feature. (The ProPattern handle is described in detail in the chapter Element Trees: Patterns on page 963.)

To delete a pattern, pass the corresponding ProPattern handle to the function ProPatternDelete().

To obtain the leader feature for a given pattern, pass a ProPattern object to the function ProPatternLeaderGet().

To obtain the header feature for a given pattern, pass a ProPattern object to the function ProPatternHeaderGet().

To access pattern information use the pattern element tree described in the chapter Element Trees: Patterns on page 963. You can access element tree information using the functions ProElement*(), described in the chapter Element Trees: Principles of Feature Creation on page 764.

Table-Driven Patterns

The Table-Driven Pattern functions have been deprecated. Use the Table Pattern feature element tree to read and manipulate table patterns. See the section Table Patterns on page 970 for more details.

Creating Local Groups

Function Introduced:

- **ProLocalGroupCreate()**

Local groups offer a way to collect several features together as if they were one feature. This functionality is particularly useful when you are creating patterns.

The function ProLocalGroupCreate() groups together features specified by an array of feature identifiers. The output of ProLocalGroupCreate() is the object ProGroup, which a typedef of a structure similar to ProFeature.

The feature identifiers passed to `ProLocalGroupCreate()` must correspond to features that possess consecutive regeneration numbers. That is, the feature identifiers can have any values, but the corresponding features must occupy a contiguous portion of the regeneration list. (To see the regeneration number of a feature, add the column `Feat #` to the model tree.)

If there are features whose regeneration numbers lie between those belonging to the features to be grouped, Creo Parametric asks the user whether these unspecified features are to be included in the group. If the user responds with `No`, the group is not created.

After you create a local group, you may want to refresh the model tree to see the changes. To refresh the model tree, call `ProTreetoolRefresh()`.

Read Access to Groups

Groups in Creo Parametric represent sets of contiguous features that act as a single feature for purposes of some operations. While the individual features can be affected by most operations individually, certain operations apply to the entire group:

- Suppress
- Delete
- Layer operations
- Patterning operations

For more information about local groups, see the *Part Modeling User's Guide*.

User-Defined Features (UDFs) are groups of features that can be stored in a file. When a UDF is placed in a new model, the features created are automatically assigned to a group.

A local group is a set of features that have been explicitly assigned to a group, for purposes of ease of modification or patterning.

Note

All the functions in this section work for both UDFs and local groups.

Each instance of a group is identified in Creo Parametric TOOLKIT as a `ProGroup` structure. This structure is the same a `ProFeature` data handle:

```
typedef struct pro_model_item {
          ProMdl  owner;
          int     id;
          ProType type;
}  ProGroup;
```

The integer `id` in this case is the id of the group header feature, which is the first feature in the group. All groups, including those in models created before release 200i2, are assigned with a group header feature upon retrieval.

The consequences of the group header feature for users of previous versions of Pro/TOOLKIT is as follows:

- Models that contain groups automatically get one extra feature in the regeneration list, of type `PRO_FEAT_GROUP_HEAD`. This changes the feature numbers of all subsequent features, including those in the group.
- Each group automatically contains one new feature in the arrays returned by Pro/TOOLKIT.
- Each group automatically gets a different leader feature (the group head feature is the leader). The leader is the first feature in the arrays returned by Pro/TOOLKIT.
- Each group pattern contains, of course, a series of groups, and each group in the pattern is similarly altered.

Finding Groups

Functions Introduced:

- **ProSolidGroupVisit()**
- **ProSolidGroupsCollect()**
- **ProFeatureGroupStatusGet()**
- **ProFeatureGroupGet()**

The function `ProSolidGroupVisit()` allows you to visit the groups in the solid model. The function `ProSolidGroupsCollect()` returns an array of the group structures.

The function `ProFeatureGroupStatusGet()` tells you if the specified feature is in a group.

The function `ProFeatureGroupGet()` returns the `ProGroup` that includes the feature.

Group Information

Functions Introduced

- **ProUdfNameGet()**
- **ProGroupIsTabledriven()**
- **ProGroupFeatureVisit()**
- **ProGroupFeaturesCollect()**

- **ProUdfDimensionVisit()**
- **ProUdfDimensionsCollect()**
- **ProUdfDimensionNameGet()**

The function `ProUdfNameGet()` returns the name of the group. For a local group, this is the name assigned upon creation. For a UDF-created group, this is the name of the UDF file. If the UDF is an instance in a UDF family table, the function also returns the instance name.

The function `ProGroupFeatureVisit()` traverses the members of the feature group. The function `ProGroupFeaturesCollect()` returns an array containing the feature handles.

The function `ProUdfDimensionVisit()` traverses the variable dimensions used in the creation of the UDF (this is only applicable to UDF-created groups). The function `ProUdfDimensionsCollect()` returns an array of the variable dimensions. The variable dimensions are the dimensions that Creo Parametric prompts for when you create the UDF.

The function `ProUdfDimensionNameGet()` returns the original dimension symbol for the variable dimension in the UDF. This symbol is different from the symbol assigned to the dimension in the group.

Note

In Creo Parametric 6.0.0.0 and later, for the function `ProUdfDimensionNameGet()`, it is mandatory to pass the input argument *udf* as `ProGroup` object type. If you pass any other object type, the function returns the `PRO_TK_BAD_CONTEXT` error.

Creating Groups

Functions Introduced:

- **ProLocalGroupCreate()**

The function `ProLocalGroupCreate()` creates a group out of a set of specified features. The features must represent a contiguous set of features in the solid model. (Refer also to Creating Local Groups on page 145).

Deleting Groups

Functions Introduced:

- **ProGroupUngroup()**
- **ProGroupUngroupPreAction()**
- **ProGroupUngroupPostAction()**

The function `ProGroupUngroup()` removes the indicated group and deletes the group header feature.

The function prototype `ProGroupUngroupPreAction()` should be used for a notification corresponding to the `ProNotifyType` `PRO_GROUP_UNGROUP_PRE`. This callback will be called just before the user ungroups an existing local group or UDF group in the user interface. If the application returns an error from this callback, ungroup activity will be prevented (thus providing a means by which UDFs or other groups may be effectively locked). If ungroup is being cancelled by the application, the application is required to provide an informational message to the user explaining this action.

The function prototype `ProGroupUngroupPostAction()` should be used for a notification corresponding to the `ProNotifyType` `PRO_GROUP_UNGROUP_POST`. This prototype provides information about the group that was just ungrouped:

- *solid*—The solid model that owns the group.
- *group_id*—The former group feature id.
- *udf_name*—The name of the UDF the group was created from.
- *feature_list*—The feature ids that were members of the group.

Updating or Replacing UDFs

This section lists operations, which you can perform on UDFs.

Functions Introduced:

- **ProUdfUpdate()**
- **ProUdfReplace()**
- **ProUdfFileIsPreCreo7()**

The function `ProUdfUpdate()` updates a dependent UDF to the latest version of the `.gph` file. The function should be able to locate the `.gph` file from within the session or by the search path. Only dependent UDFs are updated from their original definitions.

Use the function `ProUdfReplace()` to replace a UDF placement with a similar UDF provided that the two UDF's must use the same reference types. The input to the function can include data that would be used to respond to prompts shown during an interactive replacement (for items like scale, dimension display and orientation prompts).

The function `ProUdfFileIsPreCreo7()` identifies if the `.gph` file is created or modified in a release earlier than Creo Parametric 7.0.0.0. The input argument *gph_path* is the path to the `.gph` file.

If the `.gph` file is created or modified in a release earlier than Creo Parametric 7.0.0.0, the function outputs a `ProBoolean` with the value `PRO_B_TRUE`; otherwise, the value is `PRO_B_FALSE`.

Placing UDFs

Function Introduced:

- **ProUdfCreate()**

The function `ProUdfCreate()` is used to create a new group by retrieving and applying the contents of an existing UDF file. It is equivalent to the Creo Parametric command **Model ► User-Defined Feature**.

To understand this function explanation, you must have a good knowledge and understanding of the use of UDFs in Creo Parametric. PTC recommends that you read about UDFs in the *Part Modeling User's Guide*, and practice defining and using UDFs in Creo Parametric before you attempt to use this function.

When you create a UDF interactively, Creo Parametric prompts you for the information it needs to fix the properties of the resulting features. When you create a UDF from Creo Parametric TOOLKIT, you can provide some or all of this information programmatically by assembling the data structure that is the input to the function `ProUdfCreate()`.

During the call to `ProUdfCreate()`, Creo Parametric prompts you for the following:

- Any information the UDF needs that you did not provide in the input data structure
- Correct information to replace erroneous information

Such prompts are a useful way of diagnosing errors when you develop your application. This also means that, in addition to creating UDFs fully programmatically to provide automatic synthesis of model geometry, you can also use `ProUdfCreate()` to create UDFs semi-interactively. This can simplify the interactions needed to place a complex UDF, making it easier for the user and less prone to error.

Creating a UDF may require the following types of information:

- Name—The name of the UDF library to create, and the instance name, if applicable.
- Name or path—the name (or full path) of the UDF to create, and the instance name, if applicable.

- Dependency—Whether the UDF is independent of the UDF definition, or is modified by changes made to it.
- Scale—How to scale the UDF relative to the placement model.
- Variable parameters—The new values of the variable parameters allowed to be changed during UDF placement.
- Variable annotations—The new values of the variable gtol values, surface finish values and dimension tolerances allowed to be changed during UDF placement.
- Variable dimensions—The new values of the variable dimensions and pattern parameters; those whose values can be modified each time the UDF is created.
- Dimension display—Whether to show or blank non-variable dimensions created within the UDF group.
- References—The geometrical elements (surfaces, edges, datum planes, and so on) that the UDF needs to relate the features it contains to the existing model features. The elements correspond to the picks that Creo Parametric prompts you for when you create the UDF interactively (using the prompts defined when the UDF was set up). You cannot select an embedded datum as the UDF reference.
- Part intersections—If the UDF is being created in an assembly and contains features that modify existing solid geometry, you need to define which parts in the assembly are to be affected (or "intersected"), and at which level in the assembly each such intersection is to be visible.
- Orientations—If a UDF contains a feature whose direction is defined with respect to a datum plane (for example, a hole feature that uses a datum plane at its placement plane), Creo Parametric needs to know in which direction the new feature is to point (that is, on which side of the datum plane it should lie). When you create such a UDF interactively, Creo Parametric prompts you for this orientation with a flip arrow.
- Quadrants—If a UDF contains a linearly placed feature that references two datum planes to define its location (in the new model), Creo Parametric prompts you to pick the location of the new feature. This decides on which side of each datum plane the feature must lie. This choice is referred to as the "quadrant," because there are four combinations of possibilities for each linearly placed feature.
- External symbols—The parameter or dimension to use in place of a missing external symbol from a note callout or relation.
- Copied model names—If a UDF creates components in an assembly, this argument specifies the names of the new copied components that the placement creates.

The function `ProUdfCreate()` takes the following arguments:

- *solid*—The solid model (part or assembly) on which to place the UDF.
- *data*—The UDF creation data, described below.
- *asm_reference*—An external reference assembly for calculating intersections and external references.
- *options*—An array of option flags.
- *n_options*—The size of the options array.

The UDF Input Data Structure ProUdfdata

Most of the input needed by the function `ProUdfCreate()` is contained in the single `ProUdfdata` structure. This structure can be assembled using the `ProUdfdata` functions.

The options in the data structure correspond closely to the prompts Creo Parametric gives you when you create a UDF interactively. PTC strongly recommends that before you write the Creo Parametric TOOLKIT code to fill the structure, you experiment with creating the UDF interactively using Creo Parametric, noting what prompts it gives you, and use this as a guide to the information you need to provide.

Functions Introduced:

- **ProUdfdataAlloc()**
- **ProUdfdataFree()**
- **ProUdfdataNameSet()**
- **ProUdfdataPathSet()**
- **ProUdfdataInstancenameSet()**
- **ProUdfdataDependencySet()**
- **ProUdfdataScaleSet()**
- **ProUdfdataDimdisplaySet()**
- **ProUdfdataOrientationAdd()**
- **ProUdfdataQuadrantAdd()**

The function `ProUdfdataAlloc()` allocates memory for the `ProUdfdata` structure. The function `ProUdfdataFree()` frees the data structure memory.

The function `ProUdfdataNameSet()` allows you to set the name of the UDF (the root of the file name) to create and, optionally, the instance in the UDF family table to use.

Use the function `ProUdfdataPathSet()` to set the path of the UDF file. `ProUdfCreate()` will use the input from this path, if set, otherwise the data from `ProUdfdataNameSet()` is used.

Use function `ProUdfdataInstancenameSet()` to assign the instance to be used when placing this UDF.

The function `ProUdfdataDependencySet()` specified the dependency of the UDF. The choices correspond to the choices available when you create the UDF interactively. The default for this option, if not explicitly specified, is to create the group independent of the UDF definition.

The function `ProUdfdataScaleSet()` specifies how to modify the dimensions of the UDF with respect to the placement model. The choices correspond to the options presented when you create the UDF interactively. A value for a user-defined scale can also be specified by this function. The default for this option, if not explicitly specified, is to use the same size for the UDF, regardless of the units of the placement model.

The function `ProUdfdataDimdisplaySet()` specifies how to present the non-variable dimensions in the created group. These values correspond to the options presented in Creo Parametric when placing the UDF interactively. The default for this option, if not explicitly specified, is to display the dimensions normally (allowing modification).

The function `ProUdfdataOrientationAdd()` adds to an array of orientation choices. These orientation options answer the Creo Parametric prompts that propose a flip arrow (presented, for example, when using datum planes as a reference). There should be one orientation answer presented for each prompt in Creo Parametric, and the order of the options should correspond to the order as presented in Creo Parametric. If an orientation option is not provided, the value “no flip” is applied.

The function `ProUdfdataQuadrantAdd()` adds to an array of 3-dimensional points that correspond to the picks answering the Creo Parametric prompts for the feature positions. The quadrant is requested when placing a hole or a shaft with respect to two datum planes if the UDF references were also datum planes. The order of quadrants specified should correspond to the order in which Creo Parametric prompts for them when the UDF is created interactively.

Variable Parameters and Annotations

The data structure for both variable parameters and annotations is `ProUdfvarparam`.

Functions Introduced:

- **ProUdfvarparamAlloc()**
- **ProUdfdataVarparamAdd()**

- **ProUdfvarparamValueSet()**
- **ProUdfvarparamFree()**

The function `ProUdfvarparamAlloc()` allocates a UDF variable parameter or annotation structure which describes a variable parameter or annotation. The input arguments of this function are:

- *name*—Specifies the parameter name. If it represents a variable annotation, then this must be one of the standard annotation parameter names:
  - `PTC_ GTOL_PRIMARY_TOL`—gtol value
  - `PTC_ROUGHNESS_HEIGHT`—surface finish value
  - `PTC_DIM_TOL_VALUE`—dimension symmetric tolerance value
  - `PTC_DIM_UPPER_TOL_VALUE`—upper dimension tolerance
  - `PTC_DIM_LOWER_TOL_VALUE`—lower dimension tolerance
- *item*—Specifies the owner item. This item must have type and id filled out. (The owner field is ignored by Creo Parametric). The following types are allowed here:
  - `PRO_FEATURE`
  - `PRO_ANNOTATION_ELEM`

Use the function `ProUdfdataVarparamAdd()` to add information about a variable parameter assignment to the UDF data.

The function `ProUdfvarparamValueSet()` assigns the value to be used for a variable parameter or annotation value when the UDF is placed. Note: you still must add the variable parameter to the UDF data using `ProUdfdataVarparamAdd()`.

Use the function `ProUdfvarparamFree()` to free the UDF variant parameter handle.

Variable Dimensions and Pattern Parameters

The data structure for variable dimensions and pattern parameters is `ProUdfvardim`.

Functions Introduced:

- **ProUdfvardimAlloc()**
- **ProUdfdataUdfvardimAdd()**
- **ProUdfvardimValueSet()**
- **ProUdfvardimFree()**

The function `ProUdfvardimAlloc()` sets the values used to determine the variant dimension value. This function requires the following inputs:

- *dim_name*—The symbol that the dimension or pattern parameter had when the UDF was originally defined; not the prompt that the UDF uses when interactively created. To make the name easy to remember, modify the symbols of all the dimensions that you want to select to be variable before you define the UDF that you plan to create with Creo Parametric TOOLKIT.

  If you do not remember the name, find it by creating the UDF interactively in a test model and then using the Creo Parametric TOOLKIT functions `ProUdfDimensionVisit()` and `ProUdfDimensionNameGet()` on the resulting UDF.

  If you get the name wrong, `ProUdfCreate()` does not recognize the dimension and prompts the user for the value.
- *value*—The new value of the dimension or pattern parameter.
- *type*—This enumerated type takes one of the following values:
  - `PROUDFVARTYPE_DIM`—For a dimension.
  - `PROUDFVARTYPE_IPAR`—For a pattern parameter.

The function `ProUdfdataUdfvardimAdd()` adds a variant dimension data structure to the UDF creation data.

The function `ProUdfvardimValueSet()` assigns the value to be used for a variable dimension value when the UDF is placed.

Note

The variant dimension must be added to the UDF data structure using `ProUdfdatavardimAdd()` in order for it to be used during placement.

Use the function `ProUdfvardimFree()` to free the UDF variant dimension handle.

UDF References

Functions Introduced:

- **ProUdfreferenceAlloc()**
- **ProUdfdataReferenceAdd()**
- **ProUdfreferenceFree()**

The function `ProUdfreferenceAlloc()` creates a new reference data structure. The data that must be provided to allocate the structure is:

- *prompt*—The prompt defined for this reference when the UDF was originally set up. It indicates which reference this structure is providing.
- *ref_item*—A `ProSelection` object representing the geometry to use as the reference. You can allocate an embedded datum as the UDF reference. If the reference is external, the selection component path should represent the path to the owning model relative to the external reference assembly specified in the call to `ProUdfCreate()`. If this reference item refers to an annotation reference, you can pass `NULL` to make the placed annotation incomplete.
- *external*—`PRO_B_TRUE` if the reference is external, and `PRO_B_FALSE` if it is internal.
    - Internal—The referenced element belongs directly to the model that contains the UDF. For an assembly, this means that the element belongs to the top-level assembly.
    - External—The referenced element belongs to an assembly member other than the placement member.

The function `ProUdfdataReferenceAdd()` adds the reference structure to the ProUdfdata structure.

Use the function `ProUdfreferenceFree()` to free the UDF reference handle.

Assembly Intersections

The data structure used for assembly intersections is `ProUdfintersection`.

Functions Introduced:

- **ProUdfintersectionAlloc()**
- **ProUdfdataIntersectionAdd()**
- **ProUdfintersectionFree()**

The function `ProUdfintersectionAlloc()` sets the values used to determine how a UDF placed in the context of an assembly intersects the members of the assembly. This function requires the following inputs:

- *intersect_part*—The component path from either the placement assembly or the external reference assembly down to the intersected component. The external reference assembly is provided by the *asm_reference* argument to `ProUdfCreate()`.
- *visibility*—The depth of the component path into the assembly where the intersected UDF is visible. If visibility is equal to the length of the component path, the feature is visible in the part that it intersects and all assemblies and

subassemblies. If visibility is `0`, the feature is only visible in the top-level assembly.

- *instance_names*—An array of names for the new instances of parts created to represent the intersection geometry.

The function `ProUdfdataIntersectionAdd()` adds the intersection structure to the `ProUdfdata` structure.

Use the function `ProUdfintersectionFree()` to free the UDF intersection handle.

External Symbol: Parameters

The data structure for external symbol parameters is `ProUdfextparam`.

Functions Introduced:

- **ProUdfextparamAlloc()**
- **ProUdfdataExtparamAdd()**
- **ProUdfextparamFree()**

The function `ProUdfextparamAlloc()` allocates and sets a `ProUdfextparam` structure, which describes an external symbol referencing a parameter. The input arguments of this function are:

- *prompt*—The prompt for the external parameter symbol.
- *parameter*—The parameter which is used to resolve this external symbol in the placement model.

Use the function `ProUdfdataExtparamAdd()` to add information about an external symbol parameter to the UDF data. Use the function `ProUdfextparamFree()` to free the UDF external parameter handle.

External Symbol: Dimensions

The data structure for external symbol dimensions is `ProUdfextdim`.

Functions Introduced:

- **ProUdfextdimAlloc()**
- **ProUdfdataExtdimAdd()**
- **ProUdfextdimFree()**

Use the function `ProUdfextdimAlloc()` to allocate and set a structure which describes an external dimension symbol required by the UDF. The input arguments of this function are:

- *prompt*—Specifies the prompt used for this external symbol.
- *dimension*—Specifies the dimension handle to be used to resolve the external symbol in the placement model.

Use the function `ProUdfdataExtdimAdd()` to add information about a required external dimension symbol to the UDF data. Use the function `ProUdfextdimFree()` to free the UDF dimension external symbol handle.

Copied Model Names

The data structure used for specifying new component model names is `ProUdfmdlNames`.

Functions Introduced:

- **ProUdfmdlMdlnamesAlloc()**
- **ProUdfmdlNamesSet()**

The function `ProUdfmdlMdlnamesAlloc()` sets the values used to determine the names of new components created by the UDF placement. This function requires the following inputs:

- *old_name*—The old name of the component.
- *new_name*—The new name of the component to be created.

The function `ProUdfmdlNamesSet()` adds the model names structure to the `ProUdfdata` structure.

Reading UDF Properties

The functions in this section provide the ability to read the options for placement directly from a UDF file (a `.gph` file) in order for an application to decide at runtime the inputs it will use for placing a given UDF. The following functions operate on `ProUdfdata`. These functions are capable of reading properties from the UDF file so long as the UDF name or path has already been set by `ProUdfdataNameSet()` or `ProUdfdataPathSet()`.

Some of the data retrieved by the functions in this section uses the same data types as the corresponding `ProUdfdata` set functions used for placing the UDF (as listed in the earlier section). However, data that you read out of the `ProUdfdata` is not related to data that you are using to place the UDF. You must explicitly pass each piece of data to the `ProUdfdata` functions if you want the UDF to be placed with this information.

Variable Dimensions

Functions Introduced:

- **ProUdfdataVardimsGet()**
- **ProUdfvardimNameGet()**
- **ProUdfvardimPromptGet()**
- **ProUdfvardimDefaultvalueGet()**

Use the function `ProUdfdataVardimsGet()` to obtain an array of available variant dimensions that may be set when placing this UDF. You can use the function `ProUdfvardimProarrayFree()` to free this `ProArray` of variant dimensions.

Note

The handles obtained when the function `ProUdfdataVardimsGet()` is called are not automatically assigned to the UDF for placement. In order to place the UDF with a user-defined variant dimension value, you must use `ProUdfdataVardimAdd()`.

Use the function `ProUdfvardimNameGet()` to obtain the symbol of the variant dimension. This symbol of the dimension in the reference model should be used in `ProUdfvardimAlloc()`.

Use the function `ProUdfvardimPromptGet()` to obtain the prompt of the variant dimension.

Use the function `ProUdfvardimDefaultvalueGet()` to obtain the default value for the variant dimension.

Variable Parameters

Functions Introduced:

- **ProUdfdataVarparamsGet()**
- **ProUdfvarparamOwnerGet()**
- **ProUdfvarparamNameGet()**
- **ProUdfvarparamDefaultvalueGet()**
- **ProUdfvarparamProarrayFree()**

Use the function `ProUdfdataVarparamsGet()` to obtain an array of available variant parameters and/or annotation values that can optionally be set when placing this UDF. You can use the function `ProUdfvarparamProarrayFree()` to free this `ProArray` of variant items.

Note

The handles obtained when the function `ProUdfdataVarparamsGet()` is called are not automatically assigned to the UDF for placement. In order to place the UDF with a user-defined variant parameter or annotation value, you must use `ProUdfdataVarparamAdd()`.

Use the function `ProUdfvarparamOwnerGet()` to obtain the feature or annotation element that owns this variant parameter or annotation value.

Use the function `ProUdfvarparamNameGet()` to obtain the name or the symbol of the variant parameter or annotation value.

Use the function `ProUdfvarparamDefaultvalueGet()` to obtain the default value for the variant parameter or annotation value.

UDF References

Functions Introduced:

- **ProUdfdataRequiredreferencesGet()**
- **ProUdfrequiredrefPromptGet()**
- **ProUdfrequiredrefTypeGet()**
- **ProUdfrequiredrefIsannotationref()**
- **ProUdfrequiredrefFree()**
- **ProUdfrequiredrefProarrayFree()**

Use the function `ProUdfdataRequiredreferencesGet()` to obtain a list of the references required to be set for UDF placement. In order to use this function, the UDF data must have its name or path set, and Creo Parametric must be able to successfully find the `.gph` file based on this information.

Use the function `ProUdfrequiredrefPromptGet()` to obtain the reference prompt for a UDF reference.

Use the function `ProUdfrequiredrefTypeGet()` to obtain the type of item that should be supplied for a UDF reference.

Use the function `ProUdfrequiredrefIsannotationref()` to determine if the reference is an annotation reference and is allowed to be left incomplete.

You can use the function `ProUdfrequiredrefFree()` to free a required reference handle for a UDF. Use the function `ProUdfrequiredrefProarrayFree()` to free a `ProArray` of handles to the required references of a UDF.

External Symbols

Functions Introduced:

- **ProUdfdataExternalsymbolsGet()**
- **ProUdfextsymbolTypeGet()**
- **ProUdfextsymbolPromptGet()**
- **ProUdfextsymbolParametertypeGet()**
- **ProUdfextsymbolFree()**
- **ProUdfextsymbolProarrayFree()**

Use the function `ProUdfdataExternalsymbolsGet()` to obtain an array of external symbols required by this UDF. You can free a UDF external symbol handle using the function `ProUdfextsymbolFree()` and use the function `ProUdfextsymbolProarrayFree()` to free an array of external symbol handles.

Use the function `ProUdfextsymbolTypeGet()` to obtain the type of external symbol required (dimension or parameter).

Use the function `ProUdfextsymbolPromptGet()` to obtain the prompt for this external symbol.

Use the function `ProUdfextsymbolParametertypeGet()` used to obtain the expected parameter type for an external symbol, if the type is `PRO_UDFEXTSYMBOL_PARAM`.

Instance Names

Function Introduced:

- **ProUdfdataInstancenamesGet()**

Use the function `ProUdfdataInstancenamesGet()` to obtain an array of the instance names that may be used when placing this UDF. You can free this `ProArray` of instances using the function `ProArrayFree()`.

Notification on UDF Library Creation

Creo Parametric TOOLKIT provides the ability to be notified whenever a new UDF library is created or when one is modified. You can use this notification to store additional information about the UDF library file, for example, the names and values of parameters used in the UDF.

Functions Introduced:

- **ProUdfLibraryCompletePostAction()**

Use the function prototype `ProUdfLibraryCompletePostAction()` for a notification corresponding to the `ProNotifyType` `PRO_UDF_LIB_COMPLETE_POST`. This function provides the name of the newly created or modified UDF library file, and a list of all the features included in the UDF.

Note

If you modify a UDF library, which is independent and contains no reference model then no features will be included in the input to the notification.

Multibody Support in a UDF and a Copy feature

Automatic Filling of the Body Reference

In Creo Parametric 7.0.0.0, when you are prompted for a body reference, the prompt will be automatically filled with the default body in the following cases:

- When placing a UDF created in an earlier release, in a single body target model.
- When placing a UDF created in an earlier release, in a multibody target model, and when the configuration option `tk_pre_creo7_udf_body_autofill` is set to yes. Use this configuration option to automatically fill the default body during UDF placement in Creo Parametric TOOLKIT.
- In UDFs created in Creo Parametric 7.0.0.0, the body references are not automatically filled.

The following table lists the changes in the UDF functions `ProUdfCreate()` and `ProUdfdataRequiredreferencesGet()`:

Version of gph file	Bodies in target model	Value of the configuration option `tk_pre_creo7_udf_body_autofill`	`ProUdfCreate()`	`ProUdfdataRequiredreferencesGet()`
Release earlier than Creo Parametric 7.0.0.0	Single Body	No	Returns `PRO_TK_NO_ERROR`and automatically fills the UDF with the only available body	Returns `PRO_TK_NO_ERROR` and also the `body` ref/prompt
Release earlier than Creo Parametric 7.0.0.0	Single Body	Yes		Returns `PRO_TK_NO_ERROR` but does not return the `body` ref/prompt
Release earlier than Creo Parametric 7.0.0.0	Multibody	No	• 1. If body reference is not specified, returns `PRO_TK_MULTIBODY_UNSUPPORTED` and the UDF creation fails. • If correct body reference is	Returns `PRO_TK_NO_ERROR` and also the `body` ref/prompt

Version of gph file	Bodies in target model	Value of the configuration option tk_pre_creo7_udf_body_autofill	ProUdfCreate()	ProUdfdataRequiredreferencesGet()
			specified, returns PRO_TK_NO_ERROR	
Release earlier than Creo Parametric 7.0.0.0	Multibody	Yes	Returns PRO_TK_NO_ERROR and automatically fills the UDF with the default body	Returns PRO_TK_NO_ERROR but does not return the body ref/prompt
New	Single Body	No	Does not automatically fill the UDF.	Returns PRO_TK_NO_ERROR and also all the body ref/prompt
New	Single Body	Yes		
New	Multibody	No		
New	Multibody	Yes		

API Behavior for All Combinations of UDF Type, Creation and Placement

The following cases use an example of the Solid Extrude or Cut feature requiring 3 references for placement — Top, Right and Front datum planes.

Subordinate UDF Created in a release earlier than Creo Parametric 7.0.0.0

Created in a Part or Assembly	UDFs that are Retrieved and Placed	
	Assembly	Part
Assembly	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequireDreferencesGet()`—3 `ProUdfCreate()`—Success and returns `PRO_TK_NO_ERROR`	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequireDreferencesGet()`—3 `ProUdfCreate()`—Success and returns `PRO_TK_NO_ERROR`. Default body is used implicitly
Part	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `YES` `ProUdfdataRequireDreferencesGet()`—3 `ProUdfCreate()`—returns `PRO_TK_NO_ERROR`	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `YES` `ProUdfdataRequireDreferencesGet()`—3 `ProUdfCreate()`—automatically fills the reference with the default body and returns `PRO_TK_NO_ERROR`.
	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `NO` `ProUdfdataRequireDreferencesGet()`—4 `ProUdfCreate()`—returns `PRO_TK_NO_ERROR`	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `NO` `ProUdfdataRequireDreferencesGet()`—4 Single body `ProUdfCreate()`—automatically fills the reference with the default

Created in a Part or Assembly	UDFs that are Retrieved and Placed	
	Assembly	Part
	Note You need not fill the body reference. `ProUdfdataRequiredreferencesGet()` is for query purpose only.	body and returns `PRO_TK_NO_ERROR`. Multibody `ProUdfCreate()`—returns `PRO_TK_MULTIBODY_UNSUPPORTED`, if you do not specify the body reference and the UDF creation fails. Returns`PRO_TK_NO_ERROR`, if you do not specify the body reference correctly.

Stand-alone UDF with a Reference Model, Created in a Release Earlier than Creo Parametric 7.0.0.0

You can create a stand-alone UDF with a reference model in a part but not in an assembly

Created in a Part or Assembly	UDFs that are Retrieved and Placed	
	Assembly	Part
Assembly	NIL	NIL
Part	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `YES` `ProUdfdataRequiredreferencesGet()`—3 `ProUdfCreate()`—returns `PRO_TK_NO_ERROR`	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `YES` `ProUdfdataRequiredreferencesGet()`—3 `ProUdfCreate()`—automatically fill the reference with the default body and returns `PRO_TK_NO_ERROR`.

Created in a Part or Assembly	UDFs that are Retrieved and Placed	
	Assembly	**Part**
	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `NO` `ProUdfdataRequiredreferencesGet()`—4 `ProUdfCreate()`—returns `PRO_TK_NO_ERROR` **Note** You need not fill the body reference. `ProUdfdataRequiredreferencesGet()` is for query purpose only.	Value of the configuration option `tk_pre_creo7_udf_body_autofill` is `NO` `ProUdfdataRequiredreferencesGet()`—4 Single body `ProUdfCreate()`—automatically fill the reference with the default body and returns `PRO_TK_NO_ERROR`. Multibody `ProUdfCreate()`—returns `PRO_TK_MULTIBODY_UNSUPPORTED`, if you do not specify the body reference and the UDF creation fails. Returns`PRO_TK_NO_ERROR`, if you do not specify the body reference correctly.

Stand-alone UDF without Reference Model, Created in a Release Earlier than Creo Parametric 7.0.0.0

For a stand-alone UDF created without a reference model, there is no information where it was created and creating in an assembly is assumed. Therefore, body references are not appended.

Created in a Part or Assembly	UDFs that are Retrieved and Placed	
	Assembly	**Part**
Assembly or part. There is no information in the .gph file and as a result no difference from earlier releases	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequiredreferencesGet()`—3 `ProUdfCreate()`—Success and returns `PRO_TK_NO_ERROR`	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequiredreferencesGet()`—3 `ProUdfCreate()`—Success and returns `PRO_TK_NO_ERROR` Default body is used implicitly

Stand-alone UDF without Reference Model, Created in Creo Parametric 7.0.0.0

Body references for features like Extrude or Cut created in a part in Creo Parametric 7.0.0.0 are included in the feature definition and saved in the UDF. When such UDFs are retrieved in a part, you are always prompted in the user interface for the body references, and `ProUdfdataRequiredreferencesGet()` always returns the actual number of references stored in the UDF.

Created in a Part or Assembly	UDFs that are Retrieved and Placed	
	Assembly	**Part**
Assembly	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequiredreferencesGet()`—3 `ProUdfCreate()`—Success and returns `PRO_TK_NO_ERROR`.	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequiredreferencesGet()`—3 `ProUdfCreate()`—Success and returns `PRO_TK_NO_ERROR`. Default body is used implicitly.
Part	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequiredreferencesGet()`—4 or more 1 or more body references. `ProUdfCreate()`—returns `PRO_TK_NO_ERROR` **Note** You do not need to fill the body reference. `ProUdfdataRequiredreferencesGet()` is for query purpose only.	Irrespective of the value of the configuration option `tk_pre_creo7_udf_body_autofill` `ProUdfdataRequiredreferencesGet()`—4 or more 1 or more body references. `ProUdfCreate()`—Single or Mutlibody — Do not fill the body reference.

6

Core: 3D Geometry

This chapter deals with the objects and actions used to extract the geometry of a Creo Parametric solid. Because the geometry objects are closely related to each other and have a number of generic types of action in common, this chapter is organized not by object, but by types of action needed in Creo Parametric TOOLKIT.

Some of the objects and actions in this chapter also apply to assemblies. See the Assembly: Basic Assembly Access on page 1130 chapter for information on objects and actions specific to assemblies.

Geometry Objects

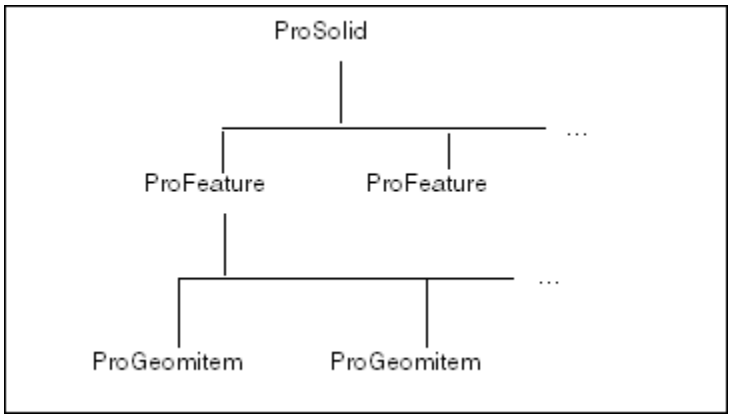
The generic object for geometry is called `ProGeomitem`, or “geometry item”. It is a DHandle that shares the declaration of `ProModelitem`. Its own instances are the specific types of geometrical item familiar to users of Creo Parametric. Each of these is declared as an OHandle, or opaque handle.

The `ProGeomitem` types are as follows:

- `ProSurface`—Surface, datum surface, or datum plane
- `ProEdge`—Edge
- `ProCurve`—Datum curve
- `ProCompcrv`—Composite datum curve
- `ProQuilt`—Quilt
- `ProAxis`—Axis
- `ProPoint`—Datum point
- `ProCsys`—Datum coordinate system

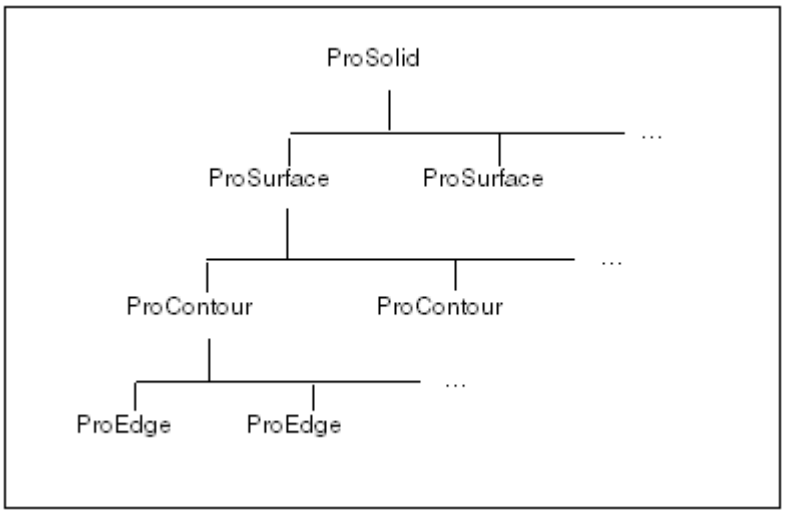
Every `ProGeomitem` is contained in a feature, and each feature is contained in a solid, as shown in the following figure.

ProGeomItem in Feature In Solid



Some geometrical items in a part are also contained in another hierarchy, which shows how they connect together geometrically, rather than to which features they belong. This type of hierarchy is shown in the following figure.

Hierarchy of Geometrical Items in a Part



The Creo Parametric TOOLKIT object `ProContour` is also an OHandle, but has no corresponding integer identifier, and therefore is not an instance of `ProGeomitem`.

There are a number of actions applicable to many of these types, whose corresponding functions begin with “ProGeomitem”. These include functions such as `ProGeomitemdataGet()`, for which there are also specific functions for the subtypes (where appropriate), and some functions for generic measurement operations. These functions are described under the context of their action type.

To read and modify the name of a `ProGeomitem`, use the functions `ProModelitemNameGet()` and `ProModelitemNameSet()`, described in the chapter Core: Models and Model Items on page 69.

Visiting Geometry Objects

Visiting geometry objects means acquiring the object handles to all the geometry objects in a solid model, either in the form of a `ProGeomitem`, or in the form of the various specific opaque handles.

The term “solid” is used in Creo Parametric TOOLKIT to distinguish models that contain three-dimensional geometry—parts and assemblies—from other model types, such as drawings. However, to the Creo Parametric user, the term “solid” is used in parts and assemblies to distinguish features that represent the geometry of the design object from features used in construction only—the various types of “datum.” Within this chapter, therefore, the terms “solid geometry” and “datums” are used in that sense.

The most general way to visit geometrical items is through their features. The section Visiting Feature Geometry on page 173 describes this in detail, and includes an illustration of the hierarchy used.

You can also traverse solid geometry items through the hierarchy of surfaces, contours, and edges in a part. This is described in the section Visiting Solid Geometry on page 175.

The following sections describe the traversal of the various datums. Some of these datums have their own visit functions, whereas others are visited through the feature hierarchy.

Note

Although the Creo Parametric user can create solid features in Assembly mode, the geometrical items that result from them are stored only within the component parts whose geometry is modified—not in the assembly features themselves. Therefore, although traversal of datums is applicable to assemblies exactly as to parts, no solid geometry items are found in assemblies.

Datum planes, datum surfaces, and solid surfaces are all represented by the `ProSurface` object because they share the same types of mathematical description.

Visiting Feature Geometry

Functions Introduced:

- **ProSolidFeatVisit()**
- **ProFeatureStatusGet()**
- **ProFeatureTypeGet()**
- **ProFeatureVisibilityGet()**
- **ProFeatureGeomitemVisit()**
- **ProGeomitemIsInactive()**
- **ProGeomitemdataGet()**

All geometry in Creo Parametric is created as a result of features, so each geometry object in Creo Parametric TOOLKIT belongs to a feature. Therefore, the most general way to traverse geometry of all types is to traverse the features, then traverse the geometry each one contains.

The function `ProSolidFeatVisit()` visits every feature in a solid. The function `ProFeatureTypeGet()` reports the type of a feature in terms of the enumerated type `ProFeattype` (described in the include file `ProFeattype.h`).

Note that ProSolidFeatVisit() is designed partly for internal use within Creo Parametric. It visits not only the features seen by the Creo Parametric users, but also the features created internally to help in the construction of geometry. These internal features are rarely of interest to Creo Parametric TOOLKIT users. To distinguish the visible features from the internal, or invisible, features, call the function ProFeatureVisibilityGet(). Internal features are invisible features used internally for construction purposes.

Note

The function ProFeatureVisibilityGet() is primarily used in the action and filter callbacks of the function ProSolidFeatVisit().

The function ProFeatureStatusGet() reports whether a feature is suppressed or inactive for some reason—only active features contain active geometry.

The function ProFeatureGeomitemVisit() visits the geometry items within a feature. It can visit all the geometry items, or one of these specific types:

SURFACE, PRO_EDGE, or PRO_CURVE. Like ProSolidFeatVisit(), this function visits not only the visible items, but also items used internally to aid in regeneration. Use the function ProGeomitemIsInactive() to skip over the internal, or inactive, geometry items. For features with solid geometry, ProFeatureGeomitemVisit() visits not only the surfaces, but also the edges. Contrast this with the visit functions specific to those items, described in the next section, that show the hierarchical relationships between surfaces, contours, and edges.

Active geometry objects for datums will usually be found in features created for them, and therefore have the corresponding type. For example, a ProGeomitem object of type PRO_CSYS is usually contained in a feature of type PRO_FEAT_CSYS. However, this is not always true; a geomitem of type PRO_AXIS can exist in a feature of type PRO_FEAT_HOLE, for example. A feature of type PRO_FEAT_MERGE, which may arise from a Mirror operation in Part mode, or from a Merge in Assembly mode, contains geometry objects corresponding to all those in the referenced features, whatever their type. In general, it is it best to make no assumptions about what kinds of feature in which you should look for datums.

Remember to distinguish the feature object from the geometry object it contains, even when they have a one-to-one relationship. For example, a feature of type PRO_FEAT_DATUM_AXIS contains a single geometry item of type PRO_AXIS, and each of these can be represented as a ProModelitem object. However, they are still distinct items with their own identifiers and types.

To extract the type and shape of each geometry item, use the function ProGeomitemdataGet(), described in detail in the section Geometry Equations on page 187.

Note

Some of the following sections about traversing specific geometry items introduce new functions specific to those types. PTC recommends that you use the more specific functions rather than the general method described in this section, because they are easier to use and usually have better performance.

All the functions in this section specific to features are described in detail in the chapter Core: Features on page 131.

Visiting Solid Geometry

Functions Introduced:

- **ProSolidBodySurfaceVisit()**
- **ProSurfaceContourVisit()**
- **ProContourEdgeVisit()**
- **ProEdgeContourGet()**
- **ProContourTraversalGet()**
- **ProContainingContourFind()**
- **ProEdgeDirGet()**
- **ProEdgeNeighborsGet()**
- **ProEdgeVertexdataGet()**

Superseded Functions:

- **ProSolidSurfaceVisit()**

In Creo Parametric 7.0.0.0 and later, the function `ProSolidSurfaceVisit()` has been deprecated. The function `ProSolidSurfaceVisit()` visits the surfaces of the model only if the model has a single body else returns the error `PRO_TK_MULTIBODY_UNSUPPORTED`.

The method `ProSolidBodySurfaceVisit()` visits all the surfaces in the specified body.

In a Creo Parametric solid, each surface contains a list of contours, and each contour contains a list of edges. The edges in a contour form a closed loop, and are ordered such that following the edges keeps the surface on the right. External contours go clockwise, and internal contours go counterclockwise.

The functions `ProSolidBodySurfaceVisit()`, `ProSurfaceContourVisit()`, and `ProContourEdgeVisit()` traverse all the objects in this three-level hierarchy. If you visit all the surfaces, the

contours of each surface, and the edges of each contour, the resulting code visits each surface and contour one time, and each edge twice. This is true because each edge forms the intersection between two surfaces, and is therefore listed in one contour of each of the two surfaces.

The function `ProEdgeContourGet()` returns a pointer to the contour, which is associated with the specified edge. The input arguments are:

- *surface*—Specifies the surface of the contour.
- *edge*—Specifies the handle of the edge.

The function `ProContourTraversalGet()` tells you whether the specified contour is internal or external. The function `ProContainingContourFind()` finds the innermost contour that closes the specified contour. If the specified contour is internal, the returned contour will be external, and vice versa. If the specified contour is the outermost contour for the surface, `ProContainingContourFind()` outputs `NULL`.

Each contour has a natural direction in terms of the order in which `ProContourEdgeVisit()` visits its edges. Each edge also has its own direction, in terms of its parameterization—the parameter, t, moves from 0 to 1 along the edge. The function `ProEdgeDirGet()` tells you whether an edge is parameterized along or against the direction of the specified contour. Note that each edge belongs to two contours, and will be in the same direction as one contour, and in the opposite direction of the other.

The function `ProEdgeNeighborsGet()` returns the two surfaces that intersect at the specified edge, and which edges on each of those surfaces is the next one following the specified edge when traversing its contour.

The function `ProEdgeVertexdataGet()` returns the list of surfaces and edges that meet at the specified vertex.

Note

The functions in this section visit active geometry items only, so you do not need to call the function `ProGeomitemIsInactive()`.

Example 1: Finding the Surfaces Penetrated by a Hole

The sample code in `UgGeomHoleSrfDisp.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_geometry` uses the techniques of feature traversal and solid geometry traversal to find the surfaces that neighbor the surfaces of the selected hole.

Visiting Axis Datums

Functions Introduced:

- **ProSolidAxisVisit()**
- **ProAxisIdGet()**
- **ProAxisInit()**
- **ProGeomitemFeatureGet()**
- **ProAxisSurfaceGet()**

An axis is represented by the object `ProAxis`, which is declared as an opaque handle. The function `ProSolidAxisVisit()` visits all the axes in a part or assembly. An axis created explicitly using the Creo Parametric command **Model ▸ Axis** will be contained in a feature of type `PRO_FEAT_DATUM_AXIS`, but axes can also exist in features of other types, such as `PRO_FEAT_HOLE`.

To find the feature that an axis belongs to, describe the axis in terms of a `ProGeomitem` object using the functions `ProAxisIdGet()` and `ProModelitemInit()`, then call `ProGeomitemFeatureGet()`.

You could also traverse axes using the functions `ProSolidFeatVisit()` and `ProFeatureGeomitemVisit()`, described in the section Visiting Feature Geometry on page 173. As input to `ProFeatureGeomitemVisit()`, use the type `PRO_AXIS`. You would have to visit features of any type that could contain axes.

Always remember to use the function `ProGeomitemIsInactive()` to skip over axes used internally only.

The function `ProAxisSurfaceGet()` provides the `ProSurface` object that identifies the surface used to define the axis position.

Visiting Coordinate System Datums

Functions Introduced:

- **ProSolidCsysVisit()**
- **ProCsysIdGet()**
- **ProCsysInit()**

A coordinate system datum is represented by the object `ProCsys`, which is declared as an opaque handle. The function `ProSolidCsysVisit()` visits all the coordinate system datums in a part or an assembly.

You could also traverse the coordinate system datums using the `ProSolidFeatVisit()` and `ProFeatureGeomitemVisit()` functions, described in the section Visiting Feature Geometry on page 173. The coordinate system datums are usually found in features of type `PRO_FEAT_CSYS`, although they can appear in others, and have the geomitem type `PRO_CSYS`.

Always remember to use the function `ProGeomitemIsInactive()` to skip over coordinate system datums used internally only.

The function `ProCsysIdGet()` provides the persistent integer identifier of the coordinate system, which is used if you need to make a `ProGeomitem` representation of the coordinate system. The function `ProCsysInit()` creates a `ProCsys` object from the integer identifier.

Visiting Datum Planes

Functions Introduced:

- **ProSurfaceInit()**
- **ProSurfaceIdGet()**

A datum plane is represented by the object `ProSurface`, which is declared as an opaque handle and is also used to represent solid surfaces and datum surfaces.

To visit all the datum planes, use the functions `ProSolidFeatVisit()` and `ProFeatureGeomitemVisit()`, described in the section Visiting Feature Geometry on page 173. The datum planes are contained in features of type `PRO_FEAT_DATUM`, each of which contains a single active `ProGeomitem` object whose `type` field is `PRO_SURFACE`.

Always remember to use the function `ProGeomitemIsInactive()` to skip over datum planes used internally only. (Active datum planes occur in features of type `PRO_FEAT_DATUM` only; datum planes created on-the-fly during creation of other features are inactive.)

To convert the `ProGeomitem` to a `ProSurface`, use the `id` field in the `ProGeomitem` as input to the function `ProSurfaceInit()`.

The function `ProSurfaceIdGet()` gives the integer identifier of a `ProSurface`, so you can convert back to a `ProGeomitem` using the function `ProModelitemInit()`.

Note

Although a datum plane has a nominal outline used to visualize the datum in the Creo Parametric display, this is not part of the geometry because a datum plane is an infinite, unbounded plane. Therefore, if you try to use the function `ProSurfaceContourVisit()` on a datum plane, it will not find any contours.

Visiting Quilts and Datum Surfaces

Functions Introduced:

- **ProSolidQuiltVisit()**
- **ProQuiltSurfaceVisit()**
- **ProQuiltIdGet()**
- **ProQuiltInit()**

A datum surface is represented by the object `ProSurface`, which is declared as an opaque handle and is also used to represent solid surfaces and datum planes.

From the viewpoint of Creo Parametric TOOLKIT, every datum surface belongs to a quilt, even if no explicit quilt feature has been made in Creo Parametric. If the user creates a single datum surface, it belongs to an internal quilt created for that purpose.

A quilt is represented by the object `ProQuilt`, which is declared as an opaque handle.

To visit datum surfaces, you must therefore first visit all the quilts, using `ProSolidQuiltVisit()`, then visit each surface in the quilt using `ProQuiltSurfaceVisit()`.

The function `ProSolidQuiltVisit()` takes the filter function `ProQuiltFilterAction()` and the visit function `ProQuiltVisitAction()` as its input arguments. The function `ProQuiltFilterAction()` is a generic action function for filtering quilts from a solid model. It returns the filter status of the quilts. This status is used as an input argument by the visit action function `ProQuiltVisitAction()`.

The function `ProQuiltSurfaceVisit()` takes the filter function `ProQuiltSurfaceFilterAction()` and the visit function `ProQuiltSurfaceVisitAction()` as its input arguments. The function `ProQuiltSurfaceFilterAction()` is a generic action function for filtering datum surfaces in a quilt. It returns the filter status of the datum surfaces. This status is used as an input argument by the visit function `ProQuiltSurfaceVisitAction()`

Always remember to use the function `ProGeomitemIsInactive()` to skip over quilts and datum surfaces used internally only.

To convert a `ProQuilt` object to a `ProGeomitem`, use the functions `ProQuiltIdGet()` and `ProModelitemInit()`.

To create a `ProQuilt` object from the integer identifier, use `ProQuiltInit()`.

To find the contours and edges of a datum surface, use the visit functions `ProSurfaceContourVisit()` and `ProContourEdgeVisit()`, described in the section Visiting Solid Geometry on page 175.

Visiting Datum Curves

Functions Introduced:

- **ProCurveIdGet()**
- **ProCurveInit()**
- **ProCurvePersistentColorGet()**
- **ProCurvePersistentColorSet()**
- **ProCurvePersistentLinestyleGet()**
- **ProCurvePersistentLinestyleSet()**

A datum curve is represented by the object `ProCurve`, which is declared as an opaque handle.

To visit all the datum curves, use the functions `ProSolidFeatVisit()` and `ProFeatureGeomitemVisit()`, described in the section Visiting Feature Geometry on page 173. The datum curves are contained in features of many different types, each of which contains one or more active `ProGeomitem` objects whose `type` field is `PRO_CURVE`.

Always remember to use the function `ProGeomitemIsInactive()` to skip over datum curves used internally only.

To convert a `ProCurve` object to a `ProGeomitem`, use the functions `ProCurveIdGet()` and `ProModelitemInit()`.

To create a `ProCurve` object from the integer identifier, use `ProCurveInit()`.

Use the functions `ProCurvePersistentColorGet()` and `ProCurvePersistentColorSet()` to obtain and set the color of a specified curve. In order to view the color changes, use the function `ProDisplistInvalidate()` on the owner model.

Use the functions `ProCurvePersistentLinestyleGet()` and `ProCurvePersistentLinestyleSet()`. In order to view the linestyle changes, use the function `ProDisplistInvalidate()` on the owner model.

Visiting Composite Datum Curves

Function Introduced:

- **ProCurveCompVisit()**

A composite datum curve is also represented by the object `ProCurve`. To distinguish a composite curve from an ordinary curve when dealing with a `ProCurve` object, use the function `ProCurveTypeGet()`. This function outputs the value `PRO_ENT_CMP_CRV` for a composite curve.

To visit all the composite datum curves, use the functions `ProSolidFeatVisit()` and`ProFeatureGeomitemVisit()`, described in the section Visiting Feature Geometry on page 173.

The composite curves are contained in features of many different types, each of which contains one or more active `ProGeomitem` objects whose `type` field is `PRO_CURVE`.

To visit the datum curves in a composite curve, use the function `ProCurveCompVisit()`.

Remember that each curve in a composite may be a composite itself, so you may need to make recursive calls. However, you can find all non-composite curves, including those contained in composites, using the method described in the previous section. It is therefore unnecessary to traverse all the composite curves to find all the non-composite curves.

Visiting Datum Points

Functions Introduced:

- **ProPointIdGet()**
- **ProPointInit()**

A datum point is represented by the object `ProPoint`, which is declared as an opaque handle.

To visit all the datum points, use the functions `ProSolidFeatVisit()` and `ProFeatureGeomitemVisit()`, described in the section Visiting Feature Geometry on page 173. The datum points are usually contained in features of type `PRO_FEAT_DATUM_POINT`, although they can also occur in others, such as `PRO_FEAT_MERGE`. Datum points are represented by geometry items of type `PRO_POINT`.

Always remember to use the function `ProGeomitemIsInactive()` to skip over datum points used internally only.

To convert a `ProPoint` object to a `ProGeomitem`, use the functions `ProPointIdGet()` and `ProModelitemInit()`.

To create a `ProPoint` object from the integer identifier, use `ProPointInit()`.

Tessellation

You can calculate tessellation for different types of Creo Parametric geometry. The tessellation is made up of small lines (for edges and curves), or triangles (for surfaces and solid models).

Curve and Edge Tessellation

Functions Introduced:

- **ProEdgeTessellationGet()**
- **ProCurveTessellationGet()**

The function `ProEdgeTessellationGet()` enables you to invoke the algorithm that generates a sequence of lines from an arbitrary curved edge. This function provides the following outputs:

- An array of the XYZ coordinates of the vertices between the tessellations
- The two surfaces that neighbor the edge (as also provided by `ProEdgeNeighborsGet()`). If the edge is a single-sided edge, then the output argument returns only one surface.
- An array of `uv` pairs for the tessellation vertices in the first neighboring surface
- An array of `uv` pairs for the second neighboring surface
- The number of tessellation vertices

The function `ProCurveTessellationGet()` retrieves the curve tessellation for a datum curve. It returns the number of tessellation points and a list of them.

Surface Tessellation

Functions Introduced:

- **ProSurfaceTessellationGet()**
- **ProTessellationFree()**
- **ProTessellationVerticesGet()**
- **ProTessellationFacetsGet()**
- **ProTessellationNormalsGet()**
- **ProTessellationParamsGet()**
- **ProSurfacetessellationinputAlloc()**
- **ProSurfacetessellationinputFree()**
- **ProSurfacetessellationinputChordheightSet()**
- **ProSurfacetessellationinputAnglecontrolSet()**
- **ProSurfacetessellationinputStepsizeSet()**
- **ProSurfacetessellationinputUvprojectionSet()**

The function `ProSurfaceTessellationGet()` calculates the tessellation data given by the `ProTessellation` object for a specified surface. Use the function `ProTessellationFree()` to release the memory used by this data object.

The function `ProTessellationVerticesGet()` obtains the vertices for the tessellation for a specified surface.

The function `ProTessellationFacetsGet()` obtains the indices indicating the vertices used for each facet of the tessellated item.

The function `ProTessellationNormalsGet()` obtains the normal vectors for each of the tessellation vertices.

The function `ProTessellationParamsGet()` obtains the UV parameters for each of the tessellation vertices.

The function `ProSurfacetessellationinputAlloc()` allocates the `ProSurfaceTessellationInput` data object containing the options for surface tessellation. Use the function `ProSurfacetessellationinputFree()` to release the memory allocated to this data object.

The function `ProSurfacetessellationinputChordheightSet()` assigns the chord height used for surface tessellation.

The function `ProSurfacetessellationinputAnglecontrolSet()` assigns the value of the angle control used for surface tessellation.

The function `ProSurfacetessellationinputStepsizeSet()` assigns the maximum value for the step size used for surface tessellation.

The function `ProSurfacetessellationinputUvprojectionSet()` assigns the parameters used to calculate the UV projection for the texture mapping to the tessellation inputs. The types of UV projection are given by the enumerated type `ProSurfaceTessellationProjection`, and are as follows:

- `PRO_SRFTESS_DEFAULT_PROJECTION`—Provides the UV parameters for the tessellation points that map to a plane whose U and V extents are [0,1] each. This is the default projection.
- `PRO_SRFTESS_PLANAR_PROJECTION`—Projects the UV parameters using a planar transform, where u=x, v=y, and z is ignored.
- `PRO_SRFTESS_CYLINDRICAL_PROJECTION`—Projects the UV parameters using a cylindrical transform, where x=r*cos(theta), y=r*sin(theta), u=theta, v=z, and r is ignored.
- `PRO_SRFTESS_SPHERICAL_PROJECTION`—Projects the UV parameters onto a sphere, where x=r*cos(theta)*sin(phi), y=r*sin(theta)*sin(phi), z=r*cos(phi), u=theta, v=phi, and r is ignored.
- `PRO_SRFTESS_NO_PROJECTION`—Provides the unmodified UV parameters for the tessellation points. This is similar to using the function `ProSurfaceParamEval()`.
- `PRO_SRFTESS_BOX_PROJECTION`—Projects the UV parameters using the box transform. The box transformation uses planar projection to project a

point from the face of the box onto the model or surface, which is opposite to the face of the box, where u = x, v = y, and z is ignored.

Note

- If the function `ProSurfacetessellationinputUvprojectionSet()` is not used, the output tessellation will not contain any UV parameters and the function `ProTessellationParamsGet()` will not return any values.
- Specify the unmodified UV parameters obtained using `PRO_SRFTESS_NO_PROJECTION` as the input `u` and `v` values for the functions `ProSurfaceXyzdataEval()`, `ProSurfaceUvpntVerify()`, `ProSurfaceDiameterEval()`, and `ProSurfacePrincipalCrvtEval()`.

Part and Assembly Tessellation

Functions Introduced:

- **ProPartTessellate()**
- **ProPartTessellationFree()**

The function `ProPartTessellate()` tessellates all surfaces of the specified part or assembly in one operation. On parts, `ProPartTessellate()` acts on all surfaces. On assemblies, this function acts only on surfaces that belong to the assembly, that is, it does not tessellate surfaces of the assembly components. `ProPartTessellate()` returns an array of `ProSurfaceTessellationData` data objects. Use the function `ProPartTessellationFree()` to release the memory assigned to these data objects.

Evaluating Geometry

The geometry of each edge or curve in Creo Parametric is described as a set of three parametric equations that represent the values of X, Y, and Z as functions of the independent parameter, `t`. For a surface, the three equations are functions of the two independent parameters `u` and `v`.

The Creo Parametric TOOLKIT functions described in this section provide the ability to evaluate the parametric edge and surface functions—that is, find the values and derivatives of X, Y and Z for the specified values of `t`, or `u` and `v`. They also provide for reverse evaluation.

Evaluating Surfaces, Edges, and Curves

Functions Introduced:

- **ProSurfaceXyzdataEval()**
- **ProEdgeXyzdataEval()**
- **ProCurveXyzdataEval()**
- **ProEdgeUvdataEval()**
- **ProSurfaceUvpntVerify()**
- **ProContourUvpntVerify()**

The function `ProSurfaceXyzdataEval()` evaluates the parametric equations for a surface at a point specified by its `u` and `v` values. The inputs to the function are the `ProSurface` object and the `u` and `v` values. The `u` and `v` values are obtained by specifying the projection type as `PRO_SRFTESS_NO_PROJECTION` for the function `ProSurfacetessellationinputUvprojectionSet()`.

The function outputs are as follows:

- The X, Y, and Z coordinates of the point, with respect to the model coordinates
- The first partial derivatives of X, Y, and Z, with respect to `u` and `v`
- The second partial derivatives of X, Y, and Z, with respect to `u` and `v`
- A unit vector in the direction of the outward normal to the surface at that point

The function `ProEdgeXyzdataEval()` performs a similar role for an edge. Its inputs are the `ProEdge` object and the value of `t` at the required point. The function outputs are as follows:

- The X, Y, and Z coordinates of the point, with respect to the model coordinates
- The first partial derivatives of X, Y, and Z, with respect to `t`
- The second partial derivatives of X, Y, and Z, with respect to `t`
- A unit vector in the direction of the edge

You must allocate a memory location for each of the output arguments of these two functions. Pass a `NULL` pointer if you do not want to use an output argument. You cannot pass a null for both the output arguments.

The function `ProCurveXyzdataEval()` is equivalent to `ProEdgeXyzdataEval()`, but works for datum curves.

The `ProEdgeUvdataEval()` function relates the geometry of a point on an edge to the surfaces that meet at that point.

The function ProSurfaceUvpntVerify() verifies whether a surface point, specified by its u and v values, lies inside, outside, or very close to the boundary of the surface. The u and v values are obtained by specifying the projection type as PRO_SRFTESS_NO_PROJECTION for the function ProSurfacetessellationinputUvprojectionSet().

Function ProContourUvpntVerify() does the same for points on a given contour.

Inverse Evaluation and Minimum Distances

Functions Introduced:

- **ProSurfaceParamEval()**
- **ProEdgeParamEval()**
- **ProCurveParamEval()**
- **ProGeomitemBodyGet()**

These functions provide the parameters of a point on a surface, edge, or datum curve nearest to the specified XYZ coordinate point.

You can use the function ProEdgeParamEval() only for the points that are either on the edge or very close to the edge.

The function ProSurfaceParamEval() returns the closest approximation to the unmodified u and v values obtained by specifying the projection type as PRO_SRFTESS_NO_PROJECTION for the function ProSurfacetessellationinputUvprojectionSet().

Use the function ProGeomitemBodyGet() to retrieve the body that is associated with the specified geometry item.

Geometry at Points

Functions Introduced:

- **ProGeometryAtPointFind()**
- **ProPoint3dOnsurfaceFind()**
- **ProPoint3dIntoleranceFind()**
- **ProSolidProjectPoint()**

The function ProGeometryAtPointFind() locates the geometry items that lie on a given point. This function supports solid geometry only.

The function `ProPoint3dOnsurfaceFind()` determines if the distance between the specified point and the specified surface is less than the Creo Parametric model accuracy as set in the current Creo Parametric session. Accuracy can also be set with function `ProSolidAccuracySet()`. This function is applicable to solid and datum surfaces.

The function `ProPoint3dIntoleranceFind()` determines if two specified points are co-incident, that is, if the distance between the two points is within the Creo Parametric tolerance set in `ProSolidToleranceGet()`.

The function `ProSolidProjectPoint()` projects a point along the shortest possible line normal to a surface, finds the point where that line hits the solid, and returns that point. Note that this function works on parts only.

Geometry Equations

Functions Introduced:

- **ProGeomitemdataGet()**
- **ProGeomitemdataFree()**

The parametric equations that describe surfaces, edges, and datum curves in Creo Parametric are documented in the Geometry Representations on page 2147 appendix. (Datum curves are geometrically equivalent to edges, but because they play a different role in Creo Parametric, they need a parallel set of functions to access them. The word curve is used as a generic word for the shape of either an edge or a datum curve.)

To know the form of a particular geometry item, you need to know not only which type of equation is being used, but also the values of the various coefficients and constants used in that equation for that item.Geometry Representations on page 2147 documents the equations using the same names for these coefficients and constants used to store them in the Creo Parametric data structures. The functions in this section enable you to get copies of the data structures containing those coefficients and constants. Therefore, you can perform your own evaluations.

The data structures for `ProSurfacedata` are defined in the include file `ProSurfacedata.h`, and those for `ProCurvedata` are defined in `ProCurvedata.h`.

The function `ProGeomitemdataGet()` allocates and fills a data structure that describes the geometry of the item. The structure `ProGeomitemdata` is declared in the file `ProGeomitemdata.h`, and looks like this:

```
typedef struct geom_item_data_struct
{
  ProType               obj_type;
  union
  {
     ProCurvedata      *p_curve_data;
```

```
      ProSurfacedata    *p_surface_data;
      ProCsysdata       *p_csys_data;
  }   data;
} ProGeomitemdata;
```

The `type` field has the same value as the `type` field in the `ProGeomitem` object.

The three fields in the union contain data structures for the geometry of curves (including solid edges and axes), surfaces, and coordinate system datums. These three data structures are described in detail in the sections Geometry of Solid Edges on page 188, Geometry of Surfaces on page 189, and Geometry of Coordinate System Datums on page 191, respectively.

The memory for the data structure is allocated by the function, but is never freed. To free the memory when you have finished with it, call `ProGeomitemdataFree()`.

Geometry of Solid Edges

Functions Introduced:

- **ProEdgeTypeGet()**
- **ProEdgeDataGet()**
- **ProEdgedataMemoryFree()**
- **ProEdgedataFree()**

Function `ProEdgeTypeGet()` provides the equation used to describe the edge. Function `ProEdgeDataGet()` returns the data structure associated with the specified edge.

Use function `ProEdgedataMemoryFree()` to free the top-level memory associated with the edge data structure. Function `ProEdgedataFree()` frees the underlying memory of the data structure.

Follow these steps to get the description of an edge:

1. Get the type of equation used to describe the edge using the function `ProEdgeTypeGet()`. The possible types for a solid edge are as follows:
   - `PRO_ENT_LINE`—A straight line
   - `PRO_ENT_ARC`—An arc
   - `PRO_ENT_ELLIPSE`—An ellipse
   - `PRO_ENT_SPLINE`—A nonuniform cubic spline
   - `PRO_ENT_B_SPLINE`—A nonuniform rational B-spline (NURBS)
2. Get the data structure for the geometry using the function `ProEdgeDataGet()`. For an edge, the `type` field is set to `PRO_EDGE`, and

the relevant field from the union is `p_curve_data`. The type for that field, `ProCurvedata`, is itself a union that contains a field for each type of edge equation. For example, if the edge type is `PRO_ENT_ARC`, the relevant field in the `ProCurvedata` structure is the one called `arc`, of type `ProArcdata`. Each such structure contains fields for the coefficients and constants in the relevant equations (described in the Geometry Representations on page 2147 appendix), and share the same names.

3. When you have read the information you need from the `ProGeomitemdata` structure, free the memory using `ProGeomitemdataFree()`.

Example 2: Extracting the Diameter of an Arc Edge

The sample code in `UgGeomArcDiaDisp.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_geometry` shows how to extract the geometry equation of a solid edge.

Geometry of Surfaces

Functions Introduced:

- **ProSurfaceTypeGet()**
- **ProSurfaceDataGet()**
- **ProSurfaceIsDatumPlane()**
- **ProSurfaceSameSrfsFind()**
- **ProSldsurfaceVolumesFind()**
- **ProSurfacePeriodicityGet()**
- **ProSurfaceNextGet()**

The method for getting the description of surface geometry is analogous to that described in the previous section for solid edges. Function `ProSurfaceTypeGet()` provides the equation used to describe the surface. Function `ProSurfaceDataGet()` returns the data structure associated with the specified surface.

Use function `ProSurfacedataMemoryFree()` to free the top-level memory associated with the surface data structure. Function `ProSurfacedataFree()` frees the underlying memory of the data structure.

The possible types of surface are as follows:

- `PRO_SRF_PLANE`—A plane
- `PRO_SRF_CYL`—A cylinder
- `PRO_SRF_CONE`—A cone
- `PRO_SRF_TORUS`—A torus

- PRO_SRF_COONS—A Coons patch
- PRO_SRF_SPL—A spline surface
- PRO_SRF_FIL—A fillet surface
- PRO_SRF_RUL—A ruled surface
- PRO_SRF_REV—A surface of revolution
- PRO_SRF_TABCYL—A tabulated cylinder
- PRO_SRF_B_SPL—A nonuniform rational B-spline (NURBS)
- PRO_SRF_CYL_SPL—A cylindrical spline surface

The relevant field in the ProGeomitemdata structure is p_surface_data, of type ProSurfacedata.

The structure ProSurfacedata contains information applicable to surfaces of all types, such as the maximum and minimum values of u and v, and of X, Y, and Z for the surface, and a union that contains a field for each type of surface geometry.

As with edges, these structures contain fields for the coefficients and constants in the relevant equations, described in the Geometry Representations on page 2147 appendix.

These functions are also applicable to datum surfaces, and to datum planes (in which the surface type will always be PRO_SRF_PLANE).

The function ProSurfaceIsDatumPlane() identifies if the given surface is a datum plane.

The function ProSurfaceSameSrfsFind() finds and returns an array of surfaces that are the same as the input surface. For example, in case of a cylinder, Creo Parametric creates two half cylindrical surfaces. If you obtain one half, the other half is returned by this function.

The function ProSldsurfaceVolumesFind() analyzes and returns the number of connect volumes of a part and the surfaces that bound them.

The function ProSurfacePeriodicityGet() gets information about the periodicity of a surface. The output arguments are:

- *periodic_in_u*—Specifies if the surface is periodic in U-direction.
- *period_in_u*—Specifies the value of period in U-direction.
- *periodic_in_v*—Specifies if the surface is periodic in V-direction.
- *period_in_v*—Specifies the value of period in V-direction.

The function ProSurfaceNextGet() returns the next surface in the surface list or returns NULL if there is no next surface. The input argument this_surface is the surface for which the next surface is queried and can be NULL.

The output argument p_next_surface is a non-Null pointer for returning the next surface. If the surface passed through the input argument this_surface is the last surface in the list, then the returned surface is NULL.

Note

For a solid body part with multiple bodies, this function does not return a surface that is outside the body.

Example 3: Getting the Angle of a Conical Surface

The sample code in UgGeomConeAngDisp.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_geometry shows how to read the geometry equation of a surface.

Geometry of Axes

Function Introduced:

- **ProAxisDataGet()**

An axis is treated in the same way as a solid edge. The function ProAxisDataGet() allocates and fills a ProGeomitemdata structure for a ProAxis object. The relevant field in the union is p_curve_data, but the type of the contained edge is always a line.

Geometry of Coordinate System Datums

Function Introduced:

- **ProCsysDataGet()**

The function ProCsysDataGet() provides a ProGeomitemdata structure in which the field p_csys_data is set. This is a pointer to a structure called ProCsysdata, declared in ProCsysdata.h, that contains the location of the origin, and the directions of the three axes, of the coordinate system datum.

Geometry of Datum Planes

Datum planes are treated exactly like surfaces, so you can use ProSurfaceDataGet().

Their type is always PRO_SRF_PLANE.

Geometry of Quilts

Functions Introduced:

- **ProQuiltdataGet()**
- **ProQuiltdataSurfArrayGet()**
- **ProQuiltdataSurfArraySet()**
- **ProQuiltdataMemoryFree()**
- **ProQuiltdataFree()**
- **ProQuiltdataTypeGet()**
- **ProQuiltdataTypeSet()**
- **ProQuiltVolumeEval()**
- **ProQuiltIsBackupgeometry()**

A quilt represents a "patchwork" of connected nonsolid surfaces. A quilt may consist of a single surface or a collection of surfaces. A quilt contains information describing the geometry of all the surfaces that compose a quilt and information on how quilt surfaces are "stitched" (joined or intersected). A part can contain several quilts. You can create or manipulate quilts using a surface feature.

To find the surfaces it contains, use `ProQuiltSurfaceVisit()` and analyze the geometry of each surface.

The function `ProQuiltdataGet()` retrieves information from the quilt data structure. The helper functions `ProQuiltdataSurfArrayGet()` and `ProQuiltdataSurfArraySet()` return or define, respectively, an array of pointers to the datum surfaces in the quilt data structure.

The function `ProQuiltdataMemoryFree()` releases the top-level memory associated with the quilt data structure. The function `ProQuiltdataFree()` releases the underlying memory of the data structure.

The function `ProQuiltdataTypeGet()` returns the type of quilt data. If the type of quilt data is a body, the output argument `p_body_or_quilt` returns `PRO_BODY` and returns `PRO_QUILT` if the type is quilt.

Use the function `ProQuiltdataTypeSet()` to set the type of quilt data as `PRO_BODY` or `PRO_QUILT`. The input arguments follow:

- `p_quilt_data`—The quilt data.
- `body_or_quilt`—Type of quilt data as `PRO_BODY` or `PRO_QUILT`.

The function `ProQuiltVolumeEval()` calculates the volume of a closed quilt.

The function ProQuiltIsBackupgeometry() identifies if the specified quilt belongs to the invisible Copy Geometry backup feature. Its input argument is a pointer to the quilt's handle of the type ProQuilt. If the quilt belongs to the invisible Copy Geometry backup feature, the function returns a ProBoolean with the value PRO_B_TRUE; otherwise, the value is PRO_B_FALSE.

Geometry of Datum Surfaces

Because the system treats datum surfaces exactly like surfaces, you can use ProSurfaceDataGet().

They can have any type of geometry.

Geometry of Datum Points

Functions Introduced:

- **ProPointCoordGet()**

The function ProPointCoordGet() provides the X, Y, and Z coordinates of the specified ProPoint object.

Geometry of Datum Curves

Functions Introduced:

- **ProCurveTypeGet()**
- **ProCurveDataGet()**
- **ProCurvedataMemoryFree()**
- **ProCurvedataFree()**

Datum curves use the same data structure as edges, with the same possible types of geometry. Because they are stored in a different location in the Creo Parametric database, they need their own functions:

- ProCurveTypeGet() is analogous to ProEdgeTypeGet().
- ProCurveDataGet() is analogous to ProEdgeDataGet().

The enumerated type ProEnttype is used to get the type of curve.

- The value PRO_ENT_CMP_CRV specifies a composite curve.
- The value PRO_ENT_PARAM_CRV specifies a parametrized curve (x(t), y(t), z(t)), where x, y, and z are user-defined functions.
- The value PRO_ENT_SRF_CRV specifies a parameterized curve (u(t), v(t)) that exists on a surface, where u and v are user-defined functions.

Use function `ProCurvedataMemoryFree()` to free the top-level memory associated with the curve data structure. Function `ProCurvedataFree()` frees the underlying memory of the data structure.

Geometry of Composite Curves

A composite curve does not have geometry of its own. Find the member curves using `ProCurveCompVisit()` and analyze their geometry in the usual way.

Ray Tracing

Functions Introduced:

- **ProSolidRayIntersectionCompute()**
- **ProSelectionDepthGet()**

The function `ProSolidRayIntersectionCompute()` finds the intersections between a ray and a solid.

The ray is defined in terms of a start location and direction vector. The intersections are described in terms of an array of `ProSelection` objects to show their context in an assembly.

The function finds intersections in both directions from the start point of the ray, and assigns each intersection a depth—the distance from the ray start point in the direction defined (intersections in the reverse direction have a negative depth). You can extract the depth of each intersection using the function `ProSelectionDepthGet()`. The intersections are ordered from the most negative depth to the most positive.

The function processes all solid surfaces and datum surfaces, but not datum planes. It also includes edges that lie within a certain critical distance, called the aperture radius, of the ray. Such an edge is shown as intersected, even if a neighboring surface is also intersected. This implies that several entries in the array may represent a single “piercing,” in geometrical terms.

The aperture radius is an optional input to the function, defined in terms of pixels. If you supply a value less than –1.0, the value is taken from the Creo Parametric configuration file option `pick_aperture_radius`. If that option is not set, the function uses the default value of 7.0.

In an assembly, each component is processed separately, so if two coincident mating faces are hit, the function records two separate intersections.

Surfaces and edges that are not displayed because they are assigned to a blanked layer are not intersected.

This function is most often used in optical analysis, when calculating the path of a ray of light through an assembly whose parts represent lenses or mirrors. In this case, you want to find the closest intersecting surface in the positive direction, then calculate the normal to the surface at that point, in assembly coordinates.

Measurement

Functions Introduced:

- **ProSurfaceAreaEval()**
- **ProContourAreaEval()**
- **ProSurfaceExtremesEval()**
- **ProSurfacePrincipalCrvtEval()**
- **ProEdgeLengthEval()**
- **ProCurveLengthEval()**
- **ProEdgeLengthT1T2Eval()**
- **ProCurveLengthT1T2Eval()**
- **ProEdgeParamByLengthEval()**
- **ProCurveParamByLengthEval()**
- **ProGeomitemDistanceEval()**
- **ProGeomitemAngleEval()**
- **ProSurfaceDiameterEval()**
- **ProGeomitemDiameterEval()**
- **ProContourBoundbox3dCompute()**
- **ProContourBoundbox2dCompute()**
- **ProSelectionWithOptionsDistanceEval()**

The function `ProSurfaceAreaEval()` evaluates the surface areas of a specified surface. It is not valid for datum planes.

Function `ProContourAreaEval()` outputs the inside surface area of a specified outer contour. Note it takes into account internal voids.

The function `ProSurfaceExtremesEval()` finds the coordinates of the face edges at the extremes in the specified direction. The accuracy of the result is limited to the accuracy of edge tessellation.

Function `ProSurfacePrincipalCrvtEval()` outputs the curvatures and directions of the specified surface at a given UV point. The `u` and `v` values are obtained by specifying the projection type as `PRO_SRFTESS_NO_PROJECTION` for the function `ProSurfacetessellationinputUvprojectionSet()`.

The function ProEdgeLengthEval() evaluates the length of a solid edge, and ProCurveLengthEval() does the same for a datum curve.

Function ProEdgeLengthT1T2Eval() finds the length of a specified edge between two parameters. Use function ProCurveLengthT1T2Eval() to do the same for a curve.

ProEdgeParamByLengthEval() finds the parameter value of the point located a given length from the specified start parameter. Use function ProCurveParamByLengthEval() to do the same for a curve.

The function ProGeomitemDistanceEval() measures the distance between two geometry items. The geometry items are expressed as ProSelection objects, so you can specify any two objects in an assembly. Each object can be an axis, plane surface, or datum point.

The function ProGeomitemAngleEval() measures the angle between two geometry items expressed as ProSelection objects. Both objects must be straight, solid edges.

The function ProSurfaceDiameterEval() measures the diameter of a surface, expressed as a ProSelection object. The surface type must be one of the following:

- Cylinder—The cylinder radius
- Torus—The distance from the axis to the generating arc
- Cone—The distance of the point specified from the axis
- Surface of revolution—The distance of the point specified from the axis

The u and v values are obtained by specifying the projection type as PRO_SRFTESS_NO_PROJECTION for the function ProSurfacetessellationinputUvprojectionSet().

Note

In the case of a sphere made by revolving an arc, the Creo Parametric command **Analysis ► Diameter** gives the real spherical diameter, whereas ProGeomitemDiameterEval() gives the distance of the specified point from the axis of revolution.

The functions ProContourBoundbox2dCompute() and ProContourBoundbox3dCompute() output a bounding box for the inside surface of the specified outer contour.

Note

- Only the `ProContourBoundbox3dCompute()` function takes into account internal voids.
- The outline returned by the function `ProContourBoundbox3dCompute()` represents the outline box used by Creo Parametric embedded algorithms, and hence it can be slightly bigger than the outline computed directly from the surface parameters.

The function `ProSelectionWithOptionsDistanceEval()` evaluates the distance between two items. You can evaluate distance between surfaces, edges, entities, vertices, curves, datums, and so on. The initial selection of the item is used to guess the type of geometry. In case of error in selecting the geometry, the output argument `p_result` is set to `-1.0` and all parameters are set to `0.0`. Error types `PRO_TK_BAD_INPUTS` and `PRO_TK_BAD_CONTEXT` are returned when `p_result` is set to `-1.0`.

The input arguments `option1` and `option2` are analogous to **Options** in the **Measure** dialog box in Creo Parametric user interface. You can specify `PRO_B_TRUE` if you want to turn on the following options for the selected items:

- If the selected item is a cylindrical surface, measures the distance from the central axis of the cylindrical surface. Specify `PRO_B_FALSE` to measure from the surface instead of the axis.
- If the selected item is an arc, measures the distance from the center of a circle or an arc-shaped curve or edge. Specify `PRO_B_FALSE` to measure from the edge instead of the center.
- If the selected item is a planar surface or a plane, extends the selected surface or plane to infinity in both directions only for the purpose of measuring distance. You can now measure the distance normal to the reference entity.
- If the selected item is linear, extends the selected straight edge or curve to infinity in both directions only for the purpose of measuring distance. You can now measure the distance normal to the reference entity.

The output arguments return the following values:

- `p_result`—Distance between the two items
- `pnt_1` and `pnt_2`—Critical point for the first and second selected items. Critical point is the actual point used for measurement.
- `param_1` and `param_2`—UV parameter of the critical point for the first and second selected items

Note

The function `ProSelectionDistanceEval()` cannot be used to evaluate distance between datum planes and axes. This function will be deprecated in a future release. Use the function `ProSelectionWithOptionsDistanceEval()` instead.

Geometry as NURBS

Functions Introduced:

- **ProSurfaceToNURBS()**
- **ProEdgeToNURBS()**
- **ProCurveToNURBS()**

A common reason for extracting the solid geometry of a Creo Parametric model is to pass it to another MCAE tool for some kind of engineering analysis. Not all of the other MCAE tools share the rich variety of geometry equation types supported by Creo Parametric, and therefore may not be able to import all the surface descriptions directly. Because many MCAE systems use nonuniform rational B-splines (NURBS) to model surfaces and edges, you frequently need to convert many or all of the Creo Parametric surface descriptions to NURB splines.

The function `ProSurfaceToNURBS()` operates on a surface of any type. The function makes an accurate approximation of the shape of the surface using a NURBS, and outputs a pointer to the structure `ProSurfacedata`. This structure contains the surface type `PTC_B_SPLSRF`, which describes the form of the NURBS.

The function `ProEdgeToNURBS()` finds a one-dimensional NURBS that approximates a Creo Parametric solid edge. The function outputs a pointer to the `ProCurvedata` union whose `b_spline` field contains the NURBS description.

The function `ProCurveToNURBS()` provides the same functionality as `ProEdgeToNURBS()`, but for a datum curve.

Both `ProSurfacedata` and `ProCurvedata` are declared in the Creo Parametric TOOLKIT header file `ProGeomitem.h`.

Interference

Functions Introduced:

- **ProFitClearanceCompute()**
- **ProFitGlobalinterferenceCompute()**
- **ProFitInterferenceCompute()**
- **ProFitInterferencevolumeCompute()**
- **ProFitInterferencevolumeDisplay()**
- **ProInterferenceDataFree()**
- **ProInterferenceInfoProarrayFree()**
- **ProVolumeInterferenceCompute()**
- **ProVolumeInterferenceDisplay()**
- **ProVolumeInterferenceBodiesGet()**
- **ProVolumeInterferenceDisplayForBody()**
- **ProVolumeInterferenceInfoArrayFree()**

The function `ProFitClearanceCompute()` computes the clearance between two objects. When the function computes clearance between two parts, it also tries to determine if there is interference between them.

Use the function `ProFitGlobalinterferenceCompute()` to compute the interference in the specified assembly. If the assembly is regenerated, the interference must be recalculated. The enumerated data type `ProFitComputeSetup` specifies the set up to compute the interference for parts or subassemblies. The valid values are:

- `PRO_FIT_PART`—Computes interference between the pairs of parts in an assembly. The interference is computed only for those pairs whose volume can be calculated. The pairs whose volume could not be calculated are removed.
- `PRO_FIT_SUB_ASSEMBLY`—Computes interference between the pairs of subassemblies in an assembly, that is, computes interference between parts of different subassemblies. The interference is computed only for pairs whose volume can be calculated. The pairs whose volume could not be calculated are removed.
- `PRO_FIT_PART_DETAILED`—Computes interference between the pairs of parts in an assembly. The interference is computed for all the pairs irrespective of whether the volume can be calculated.
- `PRO_FIT_SUB_ASSEMBLY_DETAILED`—Computes interference between the pairs of subassemblies in an assembly, that is, computes interference between parts of different subassemblies. The interference is computed for all the pairs irrespective of whether the volume can be calculated.

The function `ProFitInterferenceCompute()` returns the interference information specified between two items. In assembly mode, each item is either a component part or a solid body of a part. In Part mode, each item is a solid body of the current part. If the items are regenerated, the interferences must be recalculated. The input arguments follow:

- *sel_1*—The first part or solid body.
- *sel_2*—The second part or solid body.
- *set_facets*—The option to include facets for parts.
- *set_quilts*—The option to include quilts for parts.

Note

Set the `ProBoolean` arguments *set_facets* and *set_quilts* to `PRO_B_TRUE` to include facets or quilts in the model, respectively.

Note

The interference data obtained from the functions `ProFitGlobalinterferenceCompute()` and `ProFitInterferenceCompute()` must be passed as input to the functions `ProFitInterferencevolumeCompute()` and `ProFitInterferencevolumeDisplay()`. The interference data must not include facets or quilts. They must include information about only interfering solids.

The function `ProFitInterferenceCompute()` returns an error `PRO_TK_NOT_EXIST` if one or more items specified by either of the input arguments *sel_1* and *sel_2* could not be found or does not contain any geometry with which to compute interference.

The function returns the error `PRO_TK_GENERAL_ERROR` if the interference could not be computed.

The function `ProFitInterferencevolumeCompute()` calculates volume of interference between two specified components. Use the function `ProFitInterferencevolumeDisplay()` to display the volume of interference between the two specified components.

Use the functions `ProInterferenceDataFree()` and `ProInterferenceInfoProarrayFree()` to free the interference data obtained from the functions `ProFitInterferenceCompute()` and `ProFitGlobalinterferenceCompute()` respectively.

The function ProVolumeInterferenceCompute() calculates the volume interference between the selected closed quilt and an assembly. The output argument p_intf_infos is a pointer to the ProArray, where the interference results are stored. The interference data is returned as a ProVolumeInterferenceInfo structure.

Use the function ProVolumeInterferenceInfoArrayFree() to free the interference data obtained from the function ProVolumeInterferenceCompute(). The function is supported only in DLL mode.

The function ProVolumeInterferenceDisplay() displays the curves and surfaces that interfere with the selected quilt in the specified color or hides them. This function must be called after the interference data is computed. The input arguments are:

- *inetrf_data* — Specifies a pointer to the interference data.
- *color*— Specifies the color to use for highlighting the interference.
- *hilite*— Specifies if the curves and surfaces that interfere with the selected quilt must be displayed or hidden. Specify PRO_B_TRUE for displaying the interfering component and PRO_B_FALSE for hiding it.

Use the function ProVolumeInterferenceBodiesGet() to obtain the solid bodies of the specified component that participate in the interference with the selected closed quilt. The input argument *interf_data* specified through the structure ProVolumeInterferenceData is the pointer to the interference data for the corresponding component that interferes with the quilt. The output argument r_bodies returns an array of the bodies through the structure ProSolidBody which is allocated by the function.

Call the function ProVolumeInterferenceCompute() before using ProVolumeInterferenceBodiesGet().

The function ProVolumeInterferenceBodiesGet() returns an error PRO_TK_E_NOT_FOUND if there are no solid bodies in the interference data.

The function ProVolumeInterferenceDisplayForBody() displays or hides the curves and surfaces of the specified solid body that interfere with the quilt that is selected. The input arguments follow:

- *interf_data*— Pointer to the interference data for the corresponding component that interferes with the quilt.
- *p_body*—Pointer to the solid body in the specified component.
- *color*— Specifies the color to be used for highlighting the interference and is defined by the enumerated data type ProColortype.
- *hilite*— Specifies if the curves and surfaces that interfere with the selected quilt must be displayed or hidden. Specify PRO_B_TRUE for displaying the interference geometry of the specified body and PRO_B_FALSE for hiding it.

The function ProVolumeInterferenceDisplayForBody() returns an error PRO_TK_E_NOT_FOUND if the given body does not interfere with the quilt.

Call the function ProVolumeInterferenceInfoArrayFree() to remove the interference highlighted by the function ProVolumeInterferenceDisplay() or partially highlighted by the function ProVolumeInterferenceDisplayForBody().

Faceted Geometry

Creo Parametric allows you to build a surface CAD model on top of faceted or triangular data. Each facet is uniquely identified by a face normal and three vertices.

A facet is represented by the opaque handle ProFacet, while a set of facets is represented by the DHandle ProFacetSet. ProFacetSet has the same structure as the object ProModelitem and is defined as follows:

```
typedef struct pro_model_item
{
  ProType  type;
  int      id;
  ProMdl owner;
} ProFacetSet;
```

Visiting Facets and Facet Sets

The functions described below enable you to find all the facet sets in a model and visit the facets stored in those sets.

Functions Introduced:

- **ProSolidFacetsetVisit()**
- **ProFacetsetFacetVisit()**

The function ProSolidFacetsetVisit() visits each of the facet sets in a given model. This function takes the filter function ProFacetsetFilterAction() and the visit function ProFacetsetVisitAction() as its input arguments. The function ProFacetsetFilterAction() is a generic action function for filtering the facet sets from a model. It returns the filter status of the facet sets. This status is used as an input argument by the visit action function ProFacetsetVisitAction().

The function ProFacetsetFacetVisit() visits the facets in the faceted geometry set. This function takes the filter function ProFacetFilterAction() and the visit function ProFacetVisitAction() as its input arguments. The function

`ProFacetFilterAction()` is a generic action function for filtering a facet from the faceted geometry set. It returns the filter status of the facet. This status is used as the input argument by the visit action function `ProFacetVisitAction()`.

Accessing Facet Properties

The functions described below allow you to access the face normal and three vertices or corners of a facet. A facet vertex is given by the object `ProFacetVertex`.

Functions Introduced:

- **ProFacetVerticesGet()**
- **ProFacetverticesFree()**
- **ProFacetvertexPointGet()**
- **ProFacetNormalGet()**

The function `ProFacetVerticesGet()` obtains the vertices of a specified facet.

Use the function `ProFacetverticesFree()` to release the memory allocated for the vertices of a facet.

The function `ProFacetvertexPointGet()` returns the location of the facet vertex in the model coordinate system.

The function `ProFacetNormalGet()` returns the normal vector of a facet in the model coordinate system.

7

Core: Relations

This chapter describes how to access relations on all models and model items in Creo Parametric using the functions provided in Creo Parametric TOOLKIT.

Relations

Functions Introduced:

- **ProModelitemToRelset()**
- **ProSolidRelsetVisit()**
- **ProRelsetToModelitem()**
- **ProRelsetPostregenerationInit()**
- **ProRelsetRegenerate()**
- **ProRelsetCreate()**
- **ProRelsetDelete()**
- **ProRelsetRelationsGet()**
- **ProRelsetRelationsSet()**
- **ProRelationEvalWithUnitsRefResolve()**
- **ProRelsetUnitsSensitiveSet()**
- **ProRelsetIsUnitsSensitive()**

Superseded Functions:

- **ProRelationEvalWithUnits()**

The object `ProRelset` represents the entire set of relations on any model or model item. It is an opaque handle whose contents can be accessed only through the functions described in this section.

Creo Parametric TOOLKIT can only access the relations of the models and model item types as listed in the table below:

Model Types	Description
PRO_PART	Part
PRO_ASSEMBLY	Assembly
PRO_DRAWING	Drawing
PRO_REPORT	Report
PRO_DIAGRAM	Diagram
PRO_DWGFORM	Format
PRO_UDF	User-defined feature
PRO_FEATURE	Feature
PRO_SURFACE	Surface
PRO_EDGE	Edge
PRO_WELD_PARAMS	Weld parameters
PRO_BND_TABLE	Bend table
PRO_EXTOBJ	External objects
PRO_PATREL_FIRST_DIR	Pattern direction 1
PRO_PATREL_SECOND_DIR	Pattern direction 2

Model Types	Description
PRO_RELOBJ_QUILT	Quilt
PRO_RELOBJ_CRV	Curve
PRO_RELOBJ_COMP_CRV	Compound curve
PRO_RELOBJ_ANNOT_ELEM	Annotation Element
PRO_RELOBJ_NC_STEP_OBJECT	NC Step Table Entry
PRO_RELOBJ_NC_STEP_MODEL	NC Step Table Model

The function ProModelitemToRelset() outputs a ProRelset object that contains the set of initial relations owned by the given model item. (Note that not all model items can have relations sets associated with them—only the types listed in the table.)

Use the function ProRelsetPostregenerationInit() to initialize the post-regeneration data object ProRelset. The object contains post-regeneration relations in the specified model.

Note

According to your requirement you can pass the initial relations, or the post-regeneration relations data object, ProRelset as input to the functions ProRelsetRelationsGet(), ProRelsetRelationsSet(), ProRelsetRegenerate() and ProRelsetDelete().

To get the relations of a feature pattern, the model item type should be either PRO_PATREL_FIRST_DIR or PRO_PATREL_SECOND_DIR, and the identifier should be that of the dimension on the pattern leader that drives the pattern in that direction. To find the identifiers of the pattern dimension, use the functions described in the section Manipulating Patterns on page 144.

The function ProSolidRelsetVisit() enables you to visit all the relation sets on every model item in a model. Like other visit functions, it calls a user-supplied action function for each relation set, although there is no filter function. If the user-supplied function returns any status other than PRO_TK_NO_ERROR, visiting will stop. The model types PRO_PART, PRO_ASSEMBLY, and PRO_DRAWING are supported.

The function ProRelsetToModelitem() outputs the model item that is the owner of the specified ProRelset.

You can regenerate a relation set using the function ProRelsetRegenerate(). This function also determines whether the specified relation set is valid. If an error occurred, the function returns a status other than PRO_TK_NO_ERROR.

To create a new relation set for a model item, use the function `ProRelsetCreate()`. If a relation set already exists for that item, the function returns `PRO_TK_E_FOUND`.

To delete all the relations in a specified relation set, call the function `ProRelsetDelete()`.

The function `ProRelsetRelationsGet()` extracts the text of a set of relations described by a `ProRelset` object. This function takes two arguments: the `ProRelset` for the relation set and a preallocated expandable array. The elements of the expandable array are of type `ProLine` (wide strings).

The function `ProRelsetRelationsSet()` creates a `ProRelset` object from an expandable array of `ProLine` objects that describes the relations as text. For details of the syntax and use of relations, see the Creo Parametric help.

Note

Existing relations will be overwritten by a call to `ProRelsetRelationsSet()`.

The function `ProRelationEvalWithUnits()` evaluates a line of a relation set and outputs the resulting value in the form of a `ProParamvalue` structure. Specify the input argument `consider_units` as `true` if you want the units of the relation to be considered while evaluating the relation. In this case, the result of the relation is returned along with its unit. See the chapter Core: Parameters on page 210 for a description of this data structure. The use of special pattern relation symbols such as `memb_v` or `idx1` is not supported; instead, replace these symbols with the corresponding dimension value or number, and evaluate them individually.

The function `ProRelationEvalWithUnits()` cannot be used for referencing external symbols.

In Creo Parametric 8.0.0.0 and later, the function `ProRelationEvalWithUnits()` is deprecated. Use the function `ProRelationEvalWithUnitsRefResolve()` instead.

The function `ProRelationEvalWithUnitsRefResolve()` evaluates the expression that is specified on the right side of a relation line and returns the value in the form of `ProParamvalue` structure. Relations with symbols that are referenced by parameters or dimensions on a different model can be evaluated using the function `ProRelationEvalWithUnitsRefResolve()`. Specify the input argument `consider_units` as `true` if you want the units of the relation to be considered while evaluating the relation. In this case, the result of the relation is returned along with its unit. See the chapter Core: Parameters on page 210 for a description of this data structure. The use of special pattern relation

symbols such as `memb_v` or `idx1` is not supported; instead, replace these symbols with the corresponding dimension value or number, and evaluate them individually.

The function `ProRelsetUnitsSensitiveSet()` specifies that units must be considered while solving the specified relation. Use the function `ProRelsetIsUnitsSensitive()` to check if units must be considered while solving the relation.

Adding a Customized Function to the Relations Dialog in Creo Parametric

Functions Introduced:

- **ProRelationFunctionRegister()**
- **ProRelationReadFunction()**
- **ProRelationWriteFunction()**
- **ProRelationArgscheckFunction()**

The function `ProRelationFunctionRegister()` registers a custom function that is visible to users in the Creo Parametric relations dialog. To register a custom function you may supply the following:

- An array of expected arguments. The arguments are described by their type (double, integer, etc) and attributes indicating if the argument can be skipped when the user calls the relations function. These optional arguments must fall at the end of the argument list.
- A Boolean indicating whether or not to check argument types internally. If the Boolean is set not to check the argument types internally, Creo Parametric does not need to know the contents of the arguments array. The custom function you create must handle all the user errors in this situation.
- An arguments check function, which can be used to verify the input arguments.
- A read function, which provides the value of the function when used in the right-hand side of a relation. For example
  ```
  d12  =  ptk_user_function (0.5, 5, true, inch)
  ```
- A write function, which receives the value when the function is used in the left-hand side of the relation. For example:
  ```
  ptk_user_function (assigned_value) = 14.0;
  ```
- All the callback functions are optional and may be `NULL`.

Note

Creo Parametric TOOLKIT registered relations are valid only when the function has been registered by the application. If the application is not running or not present, models that contain user-defined relations cannot evaluate these relations. In this situation, the relations are marked as errors, however, they can be commented until needed at a later time when the relation functions are reactivated.

The function type `ProRelationReadFunction()` is called when a custom relation function is used on the right-hand side of the relation. You should output the computed value of the custom relation function which is used to complete evaluation of the relation.

The function `ProRelationWriteFunction()`, is called when a custom function is used on the left hand side of a relation. The value of the right-hand side of the relation is provided as an input. You can use this type of function to initialize properties to be stored and used by your Creo Parametric TOOLKIT application.

The function `ProRelationArgscheckFunction()` performs the argument check function for a custom relation function. Return `PRO_TK_NO_ERROR` to indicate that the arguments passed to the relation are valid. Any other error causes an error to be displayed in the Relations dialog.

The following example creates three externally defined functions to be available in the relations dialog:

`ptk_set_a` and `ptk_set_b` are used from the left-hand side of relations to initialize double values to stored "A" and "B" variables.

`ptk_eval_ax_plus_b` returns the computation of Ax+B, where x is an input to the function.

These functions are available only while the Creo Parametric TOOLKIT application is running. Models can be saved with relations referencing these functions, but these models will have relation errors if retrieved while the application is not running.

Code Example

The sample code in the file `UgCustomRelationFunction.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_param` shows how to use the "Write" function for the assignment of the parameters used for calculation.

8

Core: Parameters

This chapter describes the Creo Parametric TOOLKIT functions that give access to parameters and geometric tolerances.

Parameter Objects

The object `ProParameter` describes the contents and ownership of a parameter. *ProParameter* is a DHandle whose declaration is as follows:

```
typedef struct proparameter
{
  ProType       type;
  ProName       id;
  ProParamowner owner;
} ProParameter;
  typedef struct proparamowner
  {
    ProParamfrom   type;
    union
    {
      ProModelitem item;
      ProMdl     model;
    } who;
  } ProParamowner;
    typedef enum proparamfrom
    {
      PRM_MODEL,
      PRM_ITEM
    } ProParamfrom;
```

A structure called `ProParamvalue` is used to represent the value of a parameter. Its declaration is as follows:

```
typedef struct Pro_Param_Value
{
  ProParamvalueType     type;
  ProParamvalueValue    value;
} ProParamvalue;
  typedef enum param_value_types
  {
    PRO_PARAM_DOUBLE,
    PRO_PARAM_STRING,
    PRO_PARAM_INTEGER,
    PRO_PARAM_BOOLEAN,
    PRO_PARAM_NOTE_ID,
    PRO_PARAM_VOID
  } ProParamvalueType;
  typedef union param_value_values
  {
    double    d_val;
    int       i_val;
    short     l_val;
    ProLine   s_val;
  } ProParamvalueValue;
```

Parameter Values

Functions introduced:

- **ProParamvalueSet()**
- **ProParamvalueValueGet()**
- **ProParamvalueTypeGet()**

These three functions are utilities to help you manipulate the `ProParamvalue` structure. They do not directly affect any parameter in Creo Parametric .

The function `ProParamvalueSet()` sets the value type of a `ProParamvalue` structure, and writes a value of that type to the object.

The function `ProParamvalueTypeGet()` provides the type of a `ProParamvalue` object.

The function `ProParamvalueValueGet()` reads a value of the specified type from a `ProParamvalue` structure.

Accessing Parameters

Functions introduced:

- **ProParameterInit()**
- **ProParameterValueWithUnitsGet()**
- **ProParameterValueWithUnitsSet()**
- **ProParameterIsModified()**
- **ProParameterValueReset()**
- **ProParameterCreate()**
- **ProParameterDelete()**
- **ProParameterSelect()**
- **ProParameterTableExport()**
- **ProParameterVisit()**
- **ProParameterReorder()**
- **ProParameterToFamtableItem()**
- **ProParameterUnitsGet()**
- **ProParameterUnitsAssign()**
- **ProParameterWithUnitsCreate()**
- **ProParameterScaledvalueGet()**
- **ProParameterScaledvalueSet()**

- **ProParameterDescriptionGet()**
- **ProParameterDescriptionSet()**
- **ProParameterLockstatusGet()**
- **ProParameterLockstatusSet()**

The function `ProParameterInit()` initializes a `ProParameter` object by defining its name and owner. The owner is expressed in terms of a `ProModelitem` object, and can be a Creo Parametric model, feature, surface, or edge.

If the owner is a model, use `ProMdlToModelitem()` to create the `ProModelitem` object; in other cases, use `ProModelitemInit()`.

The function `ProParameterValueWithUnitsGet()` reads the value of a parameter specified by a `ProParameter` object into a `ProParamvalue` object provided by the application. The function also retrieves the units in which the parameter value was expressed.

The function `ProParameterValueWithUnitsSet()` sets the value of a Creo Parametric parameter identified by a `ProParameter` object to a value specified in a `ProParamvalue` structure. The parameter is expressed using the value specified for the input parameter *units*.

Note

If the input argument *units* is passed as `NULL`, then the parameter will have the same units as that of the owner model.

The `ProParameterIsModified()` function returns a boolean value that indicates whether the value of the specified parameter has been modified since the last successful regeneration of the parameter owner. This function works successfully for solid models only.

The function `ProParameterValueReset()` sets the value of a parameter to the one it had at the end of the last regeneration.

The function `ProParameterCreate()` adds a new parameter to the Creo Parametric database, and returns a valid `ProParameter` object for the new parameter. This function takes input arguments such as the `ProModelitem` object for the owner, the name, and the `ProParamvalue` structure for the value.

Note

- Model items must have a name before you create a parameter for them. Geometric items such as surfaces, edges, curves, and quilts are not named by default; use `ProModelitemNameSet()` on these items before attempting to add parameters to them.
- From Creo Parametric 1.0 onwards, you can create parameters on top of an external simplified representation assembly. However, you cannot create parameters on the extracted master assembly component.

The function `ProParameterDelete()` deletes a parameter, specified by a `ProParameter` object, from the Creo Parametric database.

The function `ProParameterSelect()` allows the user to select one or more parameters of a specified model or database item from the **Parameters** dialog box in Creo Parametric . The top model from which the parameters will be selected must be displayed in the current window. The input argument *context* allows you to select parameters by context. It takes the following values:

- `PRO_PARAMSELECT_ANY`—Specifies any parameter.
- `PRO_PARAMSELECT_MODEL`—Specifies the parameters of the top-level model.
- `PRO_PARAMSELECT_PART`—Specifies the parameters of any part.
- `PRO_PARAMSELECT_ASM`—Specifies the parameters of any assembly.
- `PRO_PARAMSELECT_FEATURE`—Specifies the parameters of any feature.
- `PRO_PARAMSELECT_EDGE`—Specifies the parameters of any edge.
- `PRO_PARAMSELECT_SURFACE`—Specifies the parameters of any surface.
- `PRO_PARAMSELECT_QUILT`—Specifies the parameters of any quilt.
- `PRO_PARAMSELECT_CURVE`—Specifies the parameters of any curve.
- `PRO_PARAMSELECT_COMPOSITE_CURVE`—Specifies the parameters of any composite curve.
- `PRO_PARAMSELECT_INHERITED`—Specifies the parameters of any inheritance feature.
- `PRO_PARAMSELECT_SKELETON`—Specifies the parameters of any skeleton.

- `PRO_PARAMSELECT_COMPONENT`—Specifies the parameters of any component.
- `PRO_PARAMSELECT_ALLOW_SUBITEM_SELECTION`—Specifies the parameters of all the subitems of the top model.

Note

The signature of `ProParameterSelect()` has changed from Pro/ENGINEER Wildfire 2.0 onward.

The function `ProParameterTableExport()` exports a file containing information from a parameter table in Creo Parametric in the CSV or TXT format. If the output type is CSV, the output file contains the columns specified by the input argument *column_list*, which is a bitmask of columns. In the CSV format, only the local parameters are exported. However, if the output type is TXT, then a default set of columns is exported. In the TXT format, all the parameters in the specified model are exported.

The function `ProParameterVisit()` visits all the parameters on a specified database item.

Note

- The parameters are returned in the order that they appear in the parameter dialog box for the database item.
- `ProParameterVisit()` does not visit mass property parameters.

The function `ProParameterReorder()` reorders the given parameter to come just after the indicated parameter.

The function `ProParameterToFamtableItem()` converts `ProParameter` objects to `ProFamtableItem` objects. You may need to call `ProParameterToFamtableItem()` after calling `ProParameterSelect()` that allows you to select parameters from a menu.

The function `ProParameterUnitsGet()` fetches the units assigned to a parameter.

The function `ProParameterUnitsAssign()` assigns the specified unit to a parameter. If the parameter already has a unit assigned to it, the function will reassign the specified unit to it. The function can reassign unit only from the same quantity type. To convert a parameter with unit to a unitless parameter, pass the input argument *units* as `NULL`.

The function ProParameterWithUnitsCreate() enables the creation of a new parameter with the assigned units. To create a parameter without units, pass the input argument *units* as NULL.

The function ProParameterScaledvalueGet() retrieves the parameter value in terms of the units of the parameter, instead of the units of the owner model.

The function ProParameterScaledvalueSet() sets the parameter value in terms of the units provided, instead of using the units of the owner model.

The function ProParameterDescriptionGet() obtains the description of the parameter. The function ProParameterDescriptionSet() assigns the description of the parameter.

The function ProParameterLockstatusGet() returns the access state of the specified parameter. Use the function ProParameterLockstatusSet() to set the access state for the specified parameter. The access state is defined in the enumerated data type ProLockstatus. The valid values are:

- PRO_PARAMLOCKSTATUS_UNLOCKED—Parameters with full access are user-defined parameters, that can be modified from any application.
- PRO_PARAMLOCKSTATUS_LIMITED—Full access parameters can be set to have limited access. Limited access parameters can be modified by user, family tables and programs. These parameters cannot be modified by relations.
- PRO_PARAMLOCKSTATUS_LOCKED—Parameters with locked access are parameters that can be locked either by an external application, or by the user. You can modify parameters locked by an external application only from within an external application. You cannot modify user-defined locked parameters from within an external application.

Notification Functions

The parameter notification functions support the parameters owned by an annotation element. These functions are call back functions and are accessible by calling the function ProNotificationSet().

- **ProParameterCreateWithUnitsPreAction()**
- **ProParameterDeletePreAction()**
- **ProParameterModifyWithUnitsPreAction()**
- **ProParameterCreatePostAction()**
- **ProParameterModifyWithUnitsPostAction()**
- **ProParameterDeleteWithUnitsPostAction()**

The notification function ProParameterCreateWithUnitsPreAction() is called before the parameter is created in the Creo Parametric user interface. This function is available by calling ProNotificationSet() with the value of the

notify type as PRO_PARAM_CREATE_W_UNITS_PRE. You can specify the units for the parameter. The function ProParameterCreatePreAction() has been superseded by ProParameterCreateWithUnitsPreAction().

The notification function ProParameterDeletePreAction() is called before the parameter is deleted. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_PARAM_DELETE_PRE.

The notification function ProParameterModifyWithUnitsPreAction() is called before a parameter is modified in the Creo Parametric user interface. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_PARAM_MODIFY_W_UNITS_PRE. The function returns the old and new units of the parameter along with the values. The function ProParameterModifyPreAction() has been superseded by ProParameterModifyWithUnitsPreAction().

You can use the PreAction functions to cancel any changes made to the parameters. If any value except PRO_TK_NO_ERROR is returned, then the change is not permitted. The application must provide appropriate messaging to the user to explain the reason for which the change was rejected.

Note

You are not permitted to cancel the parameter modification events when modifying multiple parameters as a group.

The notification function ProParameterCreatePostAction() is called after a parameter has been created. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_PARAM_CREATE_POST.

The notification function ProParameterModifyWithUnitsPostAction() is called after a parameter has been modified in the Creo Parametric user interface. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_PARAM_MODIFY_W_UNITS_POST. The function returns the old and modified units of the parameter along with the values. The function ProParameterModifyPostAction() has been superseded by ProParameterModifyWithUnitsPostAction().

The notification function ProParameterDeleteWithUnitsPostAction() is called after a parameter has been deleted in the Creo Parametric user interface. This function is available by calling ProNotificationSet() with the value of the notify

type as PRO_PARAM_DELETE_W_UNITS_POST. The function ProParameterDeletePostAction() has been superseded by ProParameterDeleteWithUnitsPostAction().

Example 1: Labeling a Feature with a String Parameter

The sample code in UgParamFeatLabel.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_param shows how to label the selected feature with the specified string parameter. The program calls ProParameterInit() to find out whether the parameter already exists. If the parameter does not exist, the function calls ProParameterCreate(); if it exists, the function calls ProParameterValueSet().

Designating Parameters Windchill Servers

Functions introduced:

- **ProParameterDesignationAdd()**
- **ProParameterDesignationVerify()**
- **ProParameterDesignationRemove()**

These functions control the designation of parameters for Windchill servers. A designated parameter will become visible within these management systems as an attribute on the owning model when it is next submitted.

The function ProParameterDesignationAdd() designates an existing parameter, referred to by its ProParameter object.

The function ProParameterDesignationRemove() removes the designation.

Note

PTC does not recommend undesignating or deleting a parameter that is already exposed to a Product Data Management system, such as Windchill PDMLink or Pro/INTRALINK.

The function ProParameterDesignationVerify() tells you whether a specified model parameter is currently designated or not.

Restricted Parameters

Restricted parameters can have either:

- an enumeration which limits values only to those specified in advance; or
- a range which limits numerical values to fall within that range.

Functions introduced:

- **ProParameterIsEnumerated()**
- **ProParameterRangeGet()**

Use the function `ProParameterIsEnumerated()` to identify if a parameter is enumerated, and provides the values that may be assigned to it. If the parameter is enumerated then it gives the `ProArray` of values that are assigned to this parameter.

Use the function `ProParameterRangeGet()` to identify if a parameter's value is restricted to a certain range. If the parameter is restricted by a range then it gives the maximum and minimum value of the parameter.

Table-Restricted Parameters

A parameter table is made up of one or more parameter table sets. Each set represents one or more parameters with their assigned values or assigned ranges. A given parameter owner (model, feature, annotation element, geometry item) can only have one parameter table set applied that creates a given parameter. In Creo Parametric TOOLKIT , a parameter table set is represented by the type `ProParamtableSet` and is made up of entries, represented by `ProParamtableEntry`. A single entry represents a parameter with an assigned value or range.

Functions introduced:

- **ProParameterTablesetGet()**
- **ProMdlParamtablesetsCollect()**
- **ProParamtablesetEntriesGet()**
- **ProParamtablesetTablepathGet()**
- **ProParamtablesetLabelGet()**
- **ProParamtablesetFree()**
- **ProParamtablesetProarrayFree()**
- **ProParamtableentryValueGet()**
- **ProParamtableentryRangeGet()**
- **ProParamtableentryNameGet()**
- **ProParamtableentryProarrayFree()**
- **ProParamtablesetApply()**
- **ProRelsetConstraintsGet()**

Use the function `ProParameterTablesetGet()` to obtain the governing parameter table set for this parameter, if it's a member of a set.

Use the function `ProMdlParamtablesetsCollect()` to obtain an array of all the table sets that are available for use in the given model. This includes all sets that are loaded from table files that are set up in this session, and any other sets that were previously stored in the model.

Use the function `ProParamtablesetFree()` to free the parameter table set and the function `ProParamtablesetProarrayFree()` to free an array of parameter table sets.

Use the function `ProParamtablesetEntriesGet()` to obtain the entries that are contained in a parameter table set.

Use the function `ProParamtablesetTablepathGet()` to obtain the name of the table that owns this parameter table set. If the set is loaded from a certain table file, this is the full path. If the set has been stored in the model directly, this is the table name.

Use the function `ProParamtablesetLabelGet()` to obtain the set label for a given parameter table set.

Use the function `ProParamtableentryNameGet()` to obtain the name for the parameter in this table set.

Use the function `ProParamtableentryValueGet()` to obtain the value for the parameter in this table set. If the parameter also has a range applied, this is the default value for the parameter. Use the function `ProParamtableentryRangeGet()` to obtain the permitted range for the parameter in this table set. The output arguments for this function are:

- *minimum*—The minimum value for this parameter as set by the parameter set.
- *maximum*—The maximum value for this parameter as set by the parameter set

You can use the function `ProParamtableentryProarrayFree()` to free an array of table entries.

The function `ProParamtablesetApply()` assigns this parameter set to the given parameter owner. Parameters used by the set are created or modified, as appropriate. The parameter values are set to the default values. This function does not regenerate the model and may fail if the parameter owner already has one or more of the set's required parameters defined which are not driven by this table.

The function `ProRelsetConstraintsGet()` obtains the constraints applied to a given relation set. Constraints may be assigned when one or more parameters of the set is governed by an external parameter file.

Driven Parameters

The functions described below provide access to the item (parameter or function) driving model parameters.

Functions Introduced:

- **ProParameterDrivertypeGet()**
- **ProParameterDrivingsymbolGet()**
- **ProParameterDrivingparamSet()**
- **ProParameterDrivingFunctionGet()**
- **ProParameterDrivingFunctionSet()**

The function `ProParameterDriverGet()` has been deprecated. Use the function `ProParameterDrivertypeGet()` instead. The function `ProParameterDrivertypeGet()` retrieves the type of operation that is driving a model parameter in the form of `ProParameterDriver` object. In assemblies, you can refer to a parameter that belongs to another model. The function `ProParameterDrivertypeGet()` returns information for such parameters, which are referenced in the current model but belong to another model. The types of drivers are as follows:

- `PRO_PARAMDRIVER_PARAM`—Specifies the parameter driving the model parameter.
- `PRO_PARAMDRIVER_FUNCTION`—Specifies the function driving the model parameter.
- `PRO_PARAMDRIVER_RELATION`—Specifies the relation driving the model parameter.

The function `ProParameterDrivingparamGet()` has been deprecated. Use the function `ProParameterDrivingsymbolGet()` instead. The function `ProParameterDrivingsymbolGet()` retrieves the driving parameter for a model parameter, if the driver type is `PRO_PARAMDRIVER_PARAM`. The function `ProParameterDrivingsymbolGet()` also returns information for parameters, which are referenced in the current model but belong to another model.

The function `ProParameterDrivingparamSet()` sets the driver type for a material parameter to the value `PRO_PARAMDRIVER_PARAM`.

The function `ProParameterDrivingFunctionGet()` obtains the driving function for a material parameter, if the driver type is `PRO_PARAMDRIVER_FUNCTION`.

The function `ProParameterDrivingFunctionSet()` sets the driver type for a material parameter to the value `PRO_PARAMDRIVER_FUNCTION`.

9

Core: Coordinate Systems and Transformations

This chapter describes the various coordinate systems used by Creo Parametric and Creo Parametric TOOLKIT, and how to transform coordinates from one to another.

Coordinate Systems

Creo Parametric and Creo Parametric TOOLKIT use the following coordinate systems:

- Solid coordinate system
- Screen coordinate system
- Window coordinate system
- Drawing coordinate system
- Drawing view coordinate system
- Assembly coordinate system
- Datum coordinate system
- Section coordinate system

The following sections describe each of these coordinate systems.

Solid Coordinate System

The solid coordinate system is the three-dimensional, Cartesian coordinate system used to describe the geometry of a Creo Parametric solid model. In a part, the solid coordinate system describes the geometry of the surfaces and edges. In an assembly, the solid coordinate system also describes the locations and orientations of the assembly members.

You can visualize the solid coordinate system in Creo Parametric by creating a coordinate system datum with the option **Model ► Coordinate System**. Distances measured in solid coordinates correspond to the values of dimensions as seen by the Creo Parametric user.

Solid coordinates are used by Creo Parametric TOOLKIT for all the functions that look at geometry, and most of the functions that draw three-dimensional graphics.

Screen Coordinate System

The screen coordinate system is a two-dimensional coordinate system that describes locations in a Creo Parametric window. This is an intermediate coordinate system after which the screen points are transformed to screen pixels. All the models are first mapped to the screen coordinate system. When the user zooms or pans the view, the screen coordinate system follows the display of the solid, so a particular point on the solid always maps to the same screen coordinate. The mapping changes only when the view orientation is changed.

Screen coordinates are used by some of the graphics functions, the mouse input functions, and all the functions that draw graphics or manipulate items on a drawing.

Window Coordinate System

The window coordinate system is similar to the screen coordinate system. After mapping the models to the screen coordinate system, they are mapped to the window coordinate before being drawn to screen pixels based on screen resolution. When pan or zoom values are applied to the coordinates in the screen coordinate system, they result in window coordinates. When an object is first displayed in a window, or the option **View ► Refit** is used, the screen and window coordinates are the same.

You can use the function `ProWindowCoordinatePixelGet()` to get the window point in pixel coordinates.

The 3D point that is projected to 2D will be visible on the screen only if it lies within the outline that is returned by the function `ProWindowPixelOutlineGet()`. If not, it will be clipped and will not be visible on the screen.

Drawing Coordinate System

The drawing coordinate system is a two-dimensional system that describes the location on a drawing relative to the bottom, left corner, and measured in drawing units. For example, on a U.S. letter-sized, landscape-format drawing sheet that uses inches, the top, right corner is (11, 8.5) in drawing coordinates.

The Creo Parametric TOOLKIT functions that manipulate drawings generally use screen coordinates.

Drawing View Coordinate System

This drawing view coordinate system is used to describe the locations of entities in a drawing view.

Assembly Coordinate System

An assembly has its own coordinate system that describes the positions and orientations of the member parts and subassemblies, and the geometry of datum features created in the assembly.

When an assembly is retrieved into memory, each member is loaded too, and continues to use its own solid coordinate system to describe its geometry.

This is important when you are analyzing the geometry of a subassembly, and want to extract or display the results relative to the coordinate system of the parent assembly.

Datum Coordinate System

A coordinate system datum can be created anywhere in any part or assembly, and represents a user-defined coordinate system. It is often a requirement in a Creo Parametric TOOLKIT application to describe geometry relative to such a datum.

Section Coordinate System

Every sketch has a coordinate system used to locate entities in that sketch. Sketches used in features will use a coordinate system different from that of the solid model.

Coordinate System Transformations

Functions Introduced:

- **ProPntTrfEval()**
- **ProVectorTrfEval()**

All coordinate systems are treated in Creo Parametric TOOLKIT as if they were three-dimensional. Therefore, a point in any of the coordinate systems described is always represented in C by the following type:

```
typedef double ProPoint3d[3]
```

Vectors are distinguished for clarity by a different, though equivalent, declaration:

```
typedef double ProVector[3]
```

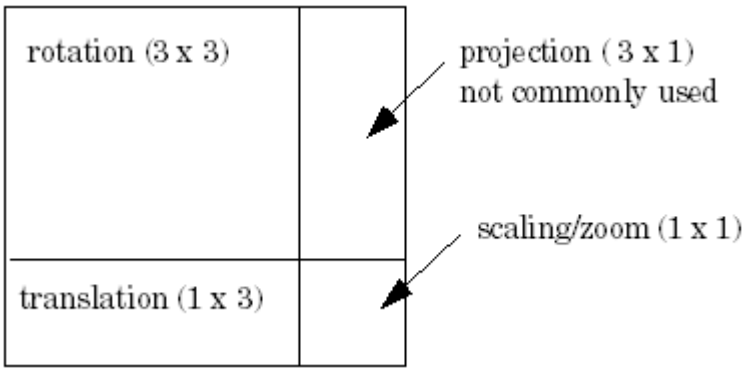
, Screen, window and section coordinates contain a Z value whose positive direction is normal to the screen or the sketch. The value of Z is not generally important when specifying a screen location as an input to a function, but it is useful in other situations. For example, if the user selects a datum plane, you can find out which side is towards the user by calculating the normal to the plane, transforming to screen coordinates, then looking at the sign of the Z coordinate.

A transformation between two coordinate systems is represented by a 4x4 matrix, with the following type:

```
typedef double ProMatrix[4][4];
```

This combines the conventional 3x3 matrix that describes the relative orientation of the two systems, and the vector that describes the shift between them.

Transformation Matrix



Creo Parametric TOOLKIT provides two utilities for performing coordinate transformations. The function `ProPntTrfEval()` transforms a three-dimensional point, and `ProVectorTrfEval()` transforms a three-dimensional vector.

The source code for other utilities that manipulate transformation matrices is located in the file `Matrix.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_examples/pt_utils/Util`.

The following sections describe the functions needed to obtain the transformation matrix between two different coordinate systems in Creo Parametric.

Transforming Solid to Screen Coordinates

Functions Introduced:

- **ProViewMatrixGet()**
- **ProViewMatrixSet()**

The view matrix describes the transformation from solid to screen coordinates. The function `ProViewMatrixGet()` provides the view matrix for the specified view. Example 1: Solid Coordinates to Screen Coordinates on page 226 shows a function that transforms a point, using the`ProViewMatrixGet()` function, and an example user function.

The function `ProViewMatrixSet()` changes the orientation of the solid model on the screen.

Example 1: Solid Coordinates to Screen Coordinates

The sample code in the file `UgFundSolid2Screen.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_fundament` shows how to transform solid coordinates to screen coordinates.

Example 2: Transform from Solid Coordinates to Screen Coordinates

The sample code in the file `UgFundCsysTrf.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_fundament` shows how to extract a selected point from the surface of a solid model, transform it from solid coordinates to screen coordinates, write the results into a file and display it.

Transforming Screen to Window Coordinates

Functions Introduced:

- **ProWindowCurrentMatrixGet()**
- **ProWindowPanZoomMatrixSet()**

Transformation from screen to window coordinates consists solely of a zoom factor and pan in X and Y.

The function `ProWindowCurrentMatrixGet()` gets the transformation matrix for the window. A pan and zoom transformation matrix consists of:

- The scale factor, running down the diagonal of the matrix. For example, to zoom in by a factor of 2, the value 2.0 will be down the diagonal in the elements (0,0), (1,1), and (2,2).
- The translation factor (pan) in the elements (3,0) - X and (3,1) - Y.
- The element at (3,3) should be 1.0.

The function `ProWindowPanZoomMatrixSet()` can change the pan and zoom of the window. The matrix should contain only the elements listed above, for function `ProWindowCurrentMatrixGet()`.

Transforming from Drawing View to Screen Coordinates in a Drawing

Function Introduced:

- **ProDrawingViewTransformGet()**

The function `ProDrawingViewTransformGet()` performs the transformation from drawing view coordinates (solid) to screen coordinates. It describes where a particular point on the solid will be in the drawing for a particular view of the solid.

Transforming from Screen to Drawing Coordinates in a Drawing

Function Introduced:

- **ProDrawingSheetTrfGet()**

The function `ProDrawingSheetTrfGet()` returns the matrix that transforms screen coordinates to drawing coordinates. The function performs this transformation for the first sheet.

Example 3: Screen Coordinates to Drawing Coordinates

The sample code in the file `UgFundScreen2Drw.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_fundament` allows you to transform the screen coordinates to drawing coordinates.

Example 4: Transform from Screen Coordinates to Drawing Coordinates

The sample code in the file `UgFundCsysTrf.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_fundament` allows you to select a point from the screen, transform it from screen coordinates to drawing coordinates, write the results into a file and display it.

Transforming Coordinates of an Assembly Member

Function Introduced:

- **ProAsmcomppathTrfGet()**

The function `ProAsmcomppathTrfGet()` provides the matrix for transforming from the solid coordinate system of the assembly member to the solid coordinates of the parent assembly, or the reverse.

Transforming to Coordinate System Datum Coordinates

Functions Introduced:

- **ProCsysDataGet()**
- **ProMatrixInit()**

The function `ProCsysDataGet()` provides the location and orientation of the coordinate system datum in the solid coordinate system of the solid that contains it. The location is in terms of the directions of the three axes, and the position of the origin. When these four vectors are made into a transformation matrix using the function `ProMatrixInit()`, that matrix defines the transformation of a point described relative to the coordinate system datum back to solid coordinates.

To transform the other way, which is the more usual requirement, you need to invert the matrix. The example function `ProUtilMatrixInvert()`, inverts the specified matrix.

Transforming Coordinates of Sketched Entities

Function Introduced:

- **ProSectionLocationGet()**

The function `ProSectionLocationGet()` provides the matrix for transforming from the solid coordinate system to the sketch coordinate system, or the reverse.

Example 5: Using Several Coordinate Transforms

The sample code in the file `UgGraphZoomAtPoint.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` demonstrates how to use several coordinate transformations. The function will zoom in on a solid model, with the results centered at the selected location.

10

Core: Family Tables

This chapter describes how to use Creo Parametric TOOLKIT functions to manipulate the family table for an object.

Family Table Objects

To enable access to family tables, Creo Parametric TOOLKIT implements the following objects (all DHandles):

- `ProFamtable`—A structure that contains the owner, type, and integer identifier of a family table.
- `ProFaminstance`—A structure that contains the name of a family table instance and the handle of the family table to which it belongs.
- `ProFamtableItem`—A structure that contains the type, name, and owner of a family table item (or column).

Family Table Utilities

Functions Introduced:

- **ProFamtableInit()**
- **ProFamtableCheck()**
- **ProFamtableEdit()**
- **ProFamtableShow()**
- **ProFamtableErase()**
- **ProFamtableIsModifiable()**

Before you can manipulate the family table information stored in an object, you must get the handle to its family table using the function `ProFamtableInit()`. Then use `ProFamtableCheck()` to determine whether the family table is empty (for a `ProSolid`, `ProPart`, or `ProAssembly` object, use `ProSolidFamtableCheck()`

The function `ProFamtableEdit()` opens a Pro/TABLE window for editing the specified family table, producing the same effect as the Creo Parametric command **Edit** in the **Family Table** dialog box.

Similarly, a call to `ProFamtableShow()` presents the family table in the same manner as the Creo Parametric command **Family Tab**, **Show**.

The function `ProFamtableErase()` clears the family from the current session, similar to the Creo Parametric command **Family Tab**, **Erase Table**.

The function `ProFamtableIsModifiable()` checks whether the specified family table can be modified.

Visiting Family Tables

Functions Introduced:

- **ProFamtableInstanceVisit()**
- **ProFamtableItemVisit()**

As with the other Creo Parametric TOOLKIT traversal functions, the traversal functions for family tables visit family table objects and pass each object to action and filter functions that you supply.

For example, the function `ProFamtableInstanceVisit()` visits all the family's instances and calls the user-supplied functions of type `ProFamtableInstanceAction()` and `ProFamtableInstanceFilter()`.

The function `ProFamtableItemVisit()` visits each family table item (or column) and calls the user-supplied functions of type `ProFamtableItemAction()` and `ProFamtableItemFilter()`.

Operations on Family Table Instances

Functions Introduced:

- **ProFaminstanceValueGet()**
- **ProFaminstanceValueSet()**
- **ProFaminstanceFamtableItemIsDefault()**
- **ProFaminstanceAdd()**
- **ProFaminstanceCheck()**
- **ProFaminstanceIsModifiable()**
- **ProFaminstanceInit()**
- **ProFaminstanceRemove()**
- **ProFaminstanceSelect()**
- **ProFaminstanceMdlGet()**
- **ProFaminstanceErase()**
- **ProFaminstanceLock()**
- **ProFaminstanceRetrieve()**
- **ProFaminstanceGenericGet()**
- **ProFaminstanceImmediategenericinfoGet()**
- **ProFaminstanceIsVerified()**
- **ProFaminstanceIsExtLocked()**
- **ProFaminstanceIsFlatState()**

The functions in this section enable you to programmatically manipulate instances that appear in a family table.

The function ProFaminstanceValueGet() retrieves the value of the family table item for the specified family table instance. Use the function ProFaminstanceValueSet() to change the value of the specified family table item.

For the specified instance, use the function ProFaminstanceFamtableItemIsDefault() to determine if the specified item has the default value, which is the value of the specified item in the generic model.

The functions ProFaminstanceValueGet(), ProFaminstanceValueSet(), and ProFaminstanceFamtableItemIsDefault() require the instance and the item handles as input values, both of which are available via the visit functions.

The function ProFaminstanceAdd() adds an instance to a family table. Note that you must initialize the handle to the new instance (using ProFaminstanceInit()) before adding the instance to the table.

Use the function ProFaminstanceRemove() to remove the specified instance from the family table.

The function ProFaminstanceMdlGet() retrieves the handle to the instance model for the given instance that is in session.

To erase an instance model from memory, call the function ProFaminstanceErase().

The function ProFaminstanceCheck() checks the existence and lock status of the specified instance. Use the function ProFaminstanceLock() to make changes to the lock status of an instance.

The function ProFaminstanceIsModifiable() checks whether the given instance of a family table can be modified.

Given the instance handle, the function ProFaminstanceRetrieve() retrieves an instance of a model from disk. Note that you must allocate space for the resulting ProMdl object. In addition, you must call ProSolidDisplay() to display the instance model.

The function ProFaminstanceGenericGet() retrieves the generic model handle for a given instance model. This function includes the ability to choose between the immediate and the top-level generic models.

From Pro/ENGINEER Wildfire 4.0 onwards, the behavior of the function ProFaminstanceGenericGet() has changed as a result of performance improvement in family table retrieval. When you now call the function ProFaminstanceGenericGet() with the flag immediate set to TRUE, the function returns a new error code PRO_TK_CANT_OPEN if the immediate generic is currently not in session. Use the function

ProFaminstanceImmediategenericinfoGet() to get the name and model type of the immediate generic model. This information can be used to retrieve the immediate generic model.

If you wish to turn-off this behavior and continue to run legacy applications in the pre-Wildfire 4.0 mode, set the configuration option retrieve_instance_dependencies to instance_and_generic_deps.

The function ProFaminstanceIsVerified() identifies whether the instance has been verified, and whether the verification succeeded or failed.

The function ProFaminstanceIsExtLocked() identifies whether the instance has been locked by an external application.

The function ProFaminstanceIsFlatState() identifies whether an instance is a sheetmetal flat state instance.

Operations on Family Table Items

Functions Introduced:

- **ProFamtableItemAdd()**
- **ProFamtableItemRemove()**
- **ProFamtableItemToModelitem()**
- **ProModelitemToFamtableItem()**
- **ProFamtableItemToParameter()**
- **ProParameterToFamtableItem()**

These functions enable you to programmatically manipulate family table items (column values).

The function ProFamtableItemAdd() adds the specified item to a family table. Similarly, ProFamtableItemRemove() removes the specified item from the family table.

The functions ProFamtableItemToModelitem() and ProModelitemToFamtableItem() convert between ProFamtableItem and ProModelitem objects. Note that user selections (ProSelection objects) can be converted to ProFamtableItem objects by calling the functions ProSelectionModelitemGet() and ProModelitemToFamtableItem().

The functions ProFamtableItemToParameter() and ProParameterToFamtableItem() convert between ProFamtableItem and ProParameter objects. Note that you might need to call ProParameterToFamtableItem() after calling ProParameterSelect() (which enables users to select parameters from a menu).

11

Core: External Data

This chapter describes how to store and retrieve external data. External data is a Creo Parametric TOOLKIT application to be stored in the Creo Parametric database in a way that is invisible to the Creo Parametric user.

Introduction to External Data

External data provides a way for the Creo Parametric TOOLKIT application to store its own private information about a Creo Parametric model within the model file. The data is built and interrogated by the Creo Parametric TOOLKIT application as a workspace data structure. It is saved to the model file when the model is saved, and retrieved when the model is retrieved. The external data is otherwise ignored by Creo Parametric, so the Creo Parametric TOOLKIT application has complete control over the form and content.

The external data for a particular Creo Parametric model is broken down into classes and slots. A class is a named “bin” for your data, and simply identifies it as yours so no other Creo Parametric TOOLKIT application (or other classes in your own application) will use it by mistake. A Creo Parametric TOOLKIT application usually needs only one class. The class name should be unique for each model, and describe the role of the data in your application.

Each class contains a list of data slots. Each slot is identified by either a name or an identifier, and contains a single data item of one of the following types:

- Integer
- Double
- Wide string (maximum length = 512 characters)
- Stream (maximum size = 512 kilobytes). A slot of type stream contains a completely unformatted sequence of bytes with unrestricted values. The slot also records the number of bytes in the stream, so no termination rules are assumed. The stream type should be used only when the format is completely controlled by your application in a platform-independent way. For example, if the volume of external data is very large, the stream format might be used to store the data in a more compressed form for greater efficiency.
- Chapter. The chapter data type is similar to the stream data. It has the following advantages as compared to stream data type:
    - Chapter data type has no limit on data length.
    - The name of the slot is used as the name of the chapter.

Stream and chapter slots could also be used as a shortcut way to store, for instance, an entire C structure, or an array of C structures, without any formatting. However, if you are supporting more than one platform with your Creo Parametric TOOLKIT application, remember that the mapping of a C structure may differ between platforms.

If external data is stored during a Creo Parametric session on one platform and retrieved on another, the values of integer, double, and wide string slots will be preserved correctly, regardless of any differences in the coding of those data types

by the two C compilers. Stream and chapter slots will be preserved with exactly the same byte values and sequence that was saved, regardless of byte-swap conventions on the two platforms.

External data is stored in the workspace and is accessible only through the functions provided for that purpose. Two objects are used to reference the data contents: `ProExtdataClass` and `ProExtdataSlot`. These are both declared as DHandles—visible data structures. The declarations are as follows:

```
typedef struct pro_extdata_class
{
  ProMdl   p_model;
  ProName  class_name;
} ProExtdataClass;

typedef struct pro_extdata_slot
{
  ProExtdataClass *p_class;
  ProName         slot_name;
  int             slot_id;
} ProExtdataSlot;
```

Each slot has two ways to be identified: a name, which is defined by the application when the slot is created, or an identifier, which is allocated automatically. You can choose which kind of identifier to use for each slot. The Creo Parametric TOOLKIT functions for external data do not use the usual return type `ProError`. Instead, they use an enumerated type called `ProExtdataErr` that contains error statuses that are more specific to the needs of those functions. All the declarations relevant to external data are in the header file `ProExtdata.h`.

Storing External Data

Functions Introduced:

- **ProExtdataInit()**
- **ProExtdataClassRegister()**
- **ProExtdataClassUnregister()**
- **ProExtdataSlotCreate()**
- **ProExtdataSlotWrite()**

- **ProExtdataSlotDelete()**
- **ProExtdataTerm()**

> **Note**
>
> For the functions `ProExtdataClassRegister()` and `ProExtdataSlotCreate()`, the combined length of the class and slot names must not exceed `PRO_NAME_SIZE`.

The first step in manipulating external data for a model in a Creo Parametric session is to call the initialize function `ProExtdataInit()` for that model.

Next, set up a class using the function `ProExtdataClassRegister()`. The inputs are the `ProMdl` object and the class name, in the form of a wide string. The function outputs a `ProExtdataClass` used thereafter by the application to reference the class.

You can delete a class that is no longer needed using the function `ProExtdataClassUnregister()`.

The function `ProExtdataSlotCreate()` creates an empty data slot. The inputs are the `ProExtdataClass` object and the slot name, in the form of a wide string. The function outputs a `ProExtdataSlot` object to identify the new slot. You can use `NULL` as the value of the slot name argument, in which case the function allocates a unique integer identifier for the slot (which becomes the value of the field `slot_id` in the `ProExtdataSlot` structure).

> **Note**
>
> Slot names cannot begin with a number.

The function `ProExtdataSlotWrite()` specifies the slot data type and writes an item of that type to the slot. The inputs are:

- The slot object `ProExtdataSlot`
- A flag showing whether the slot is identified by name or integer
- The data type of the slot
- The number of bytes in the data. Specify this argument only when the data type is stream or chapter.
- A pointer to the data (cast to `void*`)

A slot of type stream has a maximum size of 512 kilobytes. If this size is exceeded, `ProExtdataSlotWrite()` returns the status `PROEXTDATA_TK_STREAM_TOO_LARGE`. For data of size larger than 512 kilobytes, use the slot of type chapter.

You can delete an unused slot using the function `ProExtdataSlotDelete()`.

If the user and application no longer need external data in session, call `ProExtdataTerm()` to clean the external data from memory.

Note

`ProExtdataTerm()` does not affect the contents of any file on the disk. It only removes all external data from the memory. Changes made to external data during the current session are not stored in the file until you save the model. If you call `ProExtdataTerm()` after making changes to the model, all external data changes (such as newly created slots, changed slot value, and deleted slot) made since the last `ProMdlSave()` are lost.

Retrieving External Data

Functions Introduced:

- **ProExtdataLoadAll()**
- **ProExtdataClassNamesList()**
- **ProExtdataSlotIdsList()**
- **ProExtdataSlotNamesList()**
- **ProExtdataSlotRead()**
- **ProExtdataFree()**

For improved performance, external data is not loaded automatically into memory with the model. When the model is in session, call the function `ProExtdataLoadAll()` to retrieve all the external data for the specified model from the Creo Parametric model file and put it in the workspace. The function needs to be called only once to retrieve all the data.

Note that the function `ProExtdataLoadAll()` provides better performance than `ProExtdataClassNamesList()`, `ProExtdataSlotNamesList()`, and `ProExtdataSlotRead()` because these functions load only specific information (class names, slot names, and slot files, respectively), which can be slow.

The `ProExtdataClassNamesList()` function provides an array of the names of all the external data classes registered in a specified model.

The function `ProExtdataSlotIdsList()` provides an array of the integer identifiers of all the slots in a specified class. The input is a `ProExtdataClass` structure that must be set up manually or programmatically. The function `ProExtdataSlotNamesList()` provides an array of the names of the slots in the specified class. The function allocates a term in the array for each slot, even if you did not assigned a name to the slot.

The function `ProExtdataSlotRead()` reads the data type and data from a specified slot. Its input is a `ProExtdataSlot` structure that must be set up manually. There is also an input argument to show whether the slot is identified by name or by integer. The function outputs the data type, the number of bytes (if the data type is stream or chapter), and a pointer to the data itself.

The `ProExtdataSlotRead()` function allocates memory for the data it outputs. To free this memory, call `ProExtdataFree()`.

Note

If you call `ProExtdataSlotRead()` multiple times and do not free the memory, the function uses the same memory for each call.

12

Core: Cross Sections

The functions in this chapter enable you to access, modify, and delete cross sections, and create planar cross sections.

Listing Cross Sections

Functions Introduced:

- **ProXsecRename()**
- **ProXsecTypeGet()**

The function `ProXsecRename()` renames a specified cross section.

Use function `ProXsecTypeGet()` to retrieve the type of cross section. The input argument for this function is a handle to the specified cross section. The output argument *p_type* is a `ProXsecType` structure, which contains the following fields:

- *cutter*—Specifies the cutting object type. The valid cutting object types are contained in the enumerated type `ProXsecCut` and are as follows:
  - `PRO_XSEC_PLANAR`
  - `PRO_XSEC_OFFSET`
  - `PRO_XSEC_PATTERN`
- *cut_object*—Specifies the object that was cut. The valid object types are contained in the enumerated type `ProXsecCutobj` and are as follows:
  - `PRO_XSECTYPE_MODEL`—Specifies that the cross section was created on solid geometry.
  - `PRO_XSECTYPE_QUILTS`—Specifies that the cross section was created on one quilt surface.
  - `PRO_XSECTYPE_MODELQUILTS`—Specifies that the cross section was created on solid geometry and all quilt surfaces.
  - `PRO_XSECTYPE_ONEPART`—Specifies that the cross section was created on one component in the assembly.

Extracting Cross-Sectional Geometry

Functions Introduced:

- **ProXSectionItemDataGet()**
- **ProXSectionItemFree()**
- **ProXSectionItemsArrFree()**
- **ProXSectionItemsCollect()**
- **ProXsecGeometryArrayFree()**
- **ProXsecGeometryFree()**
- **ProXsecRegenerate()**

- **ProXsecDisplay()**
- **ProXsecPlaneGet()**
- **ProOffsetXsecGet()**
- **ProOffsetXsecInfoGet()**
- **ProXsecGet()**
- **ProXsecMdlnameAlloc()**
- **ProXsecMdlnameFree()**
- **ProXsecMdlnameNameGet**
- **ProXsecMdlnameNameSet()**
- **ProXsecMdlnameSolidOwnerGet()**
- **ProXsecMdlnameSolidOwnerSet()**
- **ProXsecAsModelitemGet()**
- **ProXsecFlipGet()**
- **ProXSectionExcludeCompGet()**
- **ProAsmpathProarrayFree()**

Superseded Functions:

- **ProXsecGeometryRetrieve()**
- **ProXsecGeometryCollect()**
- **ProXsecExcludeCompGet()**

The geometry of a cross section in an assembly is divided into components. Each component corresponds to one of the parts in the assembly that is intersected by the cross section, and describes the geometry of that intersection. A component can have disjoint geometry if the cross section intersects a given part instance in more than one place.

A cross section in a part has a single component.

The components of a cross section are identified by consecutive integer identifiers that always start at 0.

The function `ProXSectionItemsCollect()` returns an array of `ProXSectionItem`. An array item is created for each body. If no bodies are created, an array contains one item for each component. The function returns the error `PRO_TK_E_NOT_FOUND` when the input argument `p_view` is a drawing view and the input cross section is not found in the view.

The function `ProXSectionItemDataGet()` returns the data from the cross section body specified using the structure `ProXSectionItem`. The output arguments follow:

- `r_path`—Path to the component being cut by the cross section.
- `r_id_type`—Body or quilt type.
- `r_id`—Id of the body or the quilt being cut.
- `r_geom`—Geometry created by the cross section by cutting the specific body or quilt.

Use the function `ProXSectionItemFree()` to free the memory allocated to the `ProXSectionItem` structure.

Use the function `ProXSectionItemsArrFree()` to free the `ProArray` of cross section data allocated by the function `ProXSectionItemsCollect()`.

The function `ProXsecGeometryRetrieve()` returns an array containing the geometry of all components in the specified cross section and retrieves the following information about a specified cross-sectional component:

- The *memb_num* and *memb_id_tab* for the intersected part, with respect to the assembly that contains the cross section.
- A handle to the geometry of the intersection.

The geometry handle can be treated as an ordinary face pointer. Extract its contours with function `ProSurfaceContourVisit()`.

Use the function `ProXsecGeometryArrayFree()` to free the memory allocated to the array of cross section data and use the function `ProXsecGeometryFree()` to free memory allocated to the cross section data.

The geometry of a cross section is not maintained constantly by Creo Parametric—it is regenerated only when the user requests to see the cross section. Use function `ProXsecRegenerate()` to regenerate the cross section of a part or an assembly. Use function `ProXsecDisplay()` to display a cross section. `ProXsecDisplay()` does not add the cross section to the associated objects list, and the displayed cross section disappears on the first screen redraw.

The function `ProXsecPlaneGet()` returns the plane geometry for a specified cross section.

The function `ProOffsetXsecGet()` will be deprecated in a future release of Creo Parametric. Use the function `ProOffsetXsecInfoGet()` instead.

The function `ProOffsetXsecGet()` returns the following parameters for a specified offset cross section.

- *p_ent_arr*—Specifies a `ProArray` of `Pro2dLinedef` structures for the cross section entities.
- *plane*—Specifies an entity plane.

- *p_one_sided*—If this output argument is true, the cross section lies on one side of the entity plane, if it is false, the cross section is both-sided.
- *p_flip*—If this output argument is `false`, Creo Parametric removes material from the left of the cross section entities if the viewing direction is from the positive side of the entity plane. If *p_flip* is `true`, Creo Parametric retains the material from the left of the cross section entities and removes the rest of the material.

The function `ProOffsetXsecInfoGet()` returns the following parameters for a specified offset cross section.

- *p_ent_arr*—Specifies a `ProArray` of `Pro2dLinedef` structures for the cross section entities.
- *plane*—Specifies an entity plane.
- *p_one_sided*—If this output argument is true, the cross section lies on one side of the entity plane, if it is false, the cross section is both-sided.
- *p_flip*—If this output argument is `false`, Creo Parametric removes material from the left of the cross section entities if the viewing direction is from the positive side of the entity plane. If *p_flip* is `true`, Creo Parametric retains the material from the left of the cross section entities and removes the rest of the material.
- *p_side_xsec_mode*—Specifies the side to which the cross-section is oriented. The side is specified by the enumerated data type `ProXsecOffsetSide`. The valid values are:
  - `PRO_XSEC_OFFSET_BOTH_SIDES`—Orients the cross section to both sides of entity plane.
  - `PRO_XSEC_OFFSET_SIDE_1`—Orients the cross section to the positive normal of entity plane.
  - `PRO_XSEC_OFFSET_SIDE_2`—Orients the cross section to the negative normal of entity plane.

The function `ProXsecGet()` retrieves the cross section handle based on the specified solid model and cross section ID.

The function ProXsecMdlnameAlloc() allocates the ProXsecMdlname handle. The input arguments follow:

- *solid_owner*—Specifies the model where the cross section will be created.
- *xsec_name*—The name to set in the cross section. Maximum name size should be PRO_MDLNAME_SIZE.

Note

The function ProXsecMdlnameAlloc() returns the error PRO_TK_LINE_TOO_LONG if the name of the cross section is longer than PRO_NAME_SIZE.

Use the function ProXsecMdlnameFree() to free the memory allocated to the ProXsecMdlname handle.

The function ProXsecMdlnameNameGet() returns the name of the cross section handle.

Use the function ProXsecMdlnameNameSet() to set the name of the cross section. Maximum name size should be PRO_NAME_SIZE.

Note

The function ProXsecMdlnameNameSet() returns the error PRO_TK_LINE_TOO_LONG if the name of the cross section is longer than PRO_NAME_SIZE.

The functions ProXsecMdlnameSolidOwnerGet() and ProXsecMdlnameSolidOwnerSet() get and set the owner of the cross section.

The function ProXsecAsModelitemGet() converts a cross section handle into an appropriate model item.

The function ProXsecFlipGet() returns a integer value that indicates the direction in which the cross section has been clipped. Depending on the type of cross section, the integer value indicates different direction of clipping as below:

- Planar cross section—The integer value:
  - 1 indicates that the cross section has been clipped in the direction of the positive normal to the cross section plane.

    - `-1` indicates that the cross section has been clipped in the opposite direction of the positive normal.
- Offset cross section—The integer value:
    - `1` indicates that material has been removed from the left of the cross section entities if the viewing direction is from the positive side of the entity plane.
    - `-1` indicates that the material has been retained from the left of the cross section entities and rest of the material has been removed.

In Creo Parametric 7.0.0.0 the function `ProXsecExcludeCompGet()` is deprecated. Use the function `ProXSectionExcludeCompGet()` instead.

The function `ProXSectionExcludeCompGet()` returns the status and an array of paths to the assembly components and bodies that have been included and excluded for the specified cross section. The assembly paths are returned as a `ProArray` of type `ProSelection`. The status of the assembly components and bodies is returned by the enumerated type `ProXsecExcludeModels` and the valid values are:

- `PRO_XSEC_MODEL_EXCLUDE`—Specifies that the assembly components and bodies have been excluded.
- `PRO_XSEC_MODEL_INCLUDE`—Specifies that the assembly components and bodies have been included.

Use the function `ProSelectionarrayFree()` to free the memory allocated to the `ProArray` of type `ProSelection`.

Visiting Cross Sections

Function Introduced:

- **ProSolidXsecVisit()**
- **ProSolidXsecVisitAction()**

The function `ProSolidXsecVisit()` enables you to visit all named cross sections in the specified solid. Use `ProSolidXsecVisitAction()` to supply the function to be performed when visiting part or assembly cross sections.

Creating and Modifying Cross Sections

Functions Introduced:

- **ProXsecParallelCreate()**
- **ProXSectionOffsetCreate()**
- **ProXSectionPlanarCreate()**
- **ProXSectionCreateDataAlloc()**

- **ProXSectionCreateDataFree()**
- **ProXSectionCreateDataQuiltSelSet()**
- **ProXSectionCreateDataQuiltSelGet()**
- **ProXSectionCreateDataQuiltTypeSet()**
- **ProXSectionCreateDataQuiltTypeGet()**
- **ProXsecMakeVisible()**
- **ProXsecIsVisible()**
- **ProXsecActiveSet()**
- **ProXsecActiveGet()**
- **ProXsecCanCreateAsFeature()**
- **ProXsecOldToNewConvert()**
- **ProXsecIsFeature()**
- **ProXsecFeatureGet()**
- **ProXsecDelete()**

Superseded Functions:

- **ProXsecOffsetCreate()**
- **ProXsecPlanarWithoptionsCreate()**

The function `ProXsecParallelCreate()` creates a cross section feature parallel to a given plane.

In Creo Parametric 7.0.0.0 the function `ProXsecPlanarWithoptionsCreate()` is deprecated. Use the function `ProXSectionPlanarCreate()` instead.

The function `ProXSectionPlanarCreate()` creates a cross section feature through a datum plane and also makes the cross section visible. The input arguments are:

- *solid_owner*—Specifies the model where the cross section will be created.
- *xsec_name*—Specifies the name of the cross section.
- *cutting_plane*—Specifies the selection of the cutting plane. The cutting plane must belong to the top-level part or assembly.
- *xsec_type*—Specifies the type of object that will be cut by the cross section. It is specified by the enumerated type `ProXsecCutobj`.
- *quilt_or_one_part*—Specifies the selection of the quilt or component depending on type of object specified by *xsec_type*.

- *flip*—Specifies the direction in which the cross section will be clipped. The value 1 indicates that the cross section will be clipped in the direction of the positive normal to the cutting plane. -1 indicates that the cross section will be clipped in the opposite direction of the positive normal.
- *excld_incld_opt*—Specifies the items to exclude, specified by the input parameter *exclude_items* from cutting by the cross section and is defined by the enumerated data type `ProXsecExcludeModels`.
- *exclude_items*—Specifies a `ProArray` of selected bodies or parts to be included or excludes from the cross section.
- *data*—Reserved for future use.

Note

- From Creo Parametric 4.0 F000 onward, when a cross section is created, it is not displayed by default in the model. You must call the function `ProXsecMakeVisible()` to display the cross section.
- While porting Creo Parametric TOOLKIT applications, which have used the function `ProXsecPlanarWithoptionsCreate()` and have been created in releases prior to Creo Parametric 4.0 F000, depending on whether you want the cross section to be displayed, call the function `ProXsecMakeVisible()` in your applications. `ProXsecMakeVisible()` displays the cross section in the model.
- From Creo Parametric 2.0 onward:
  - the legacy cross sections, that is, the cross sections created in Pro/ENGINEER, Creo Elements/Pro, and in releases prior to Creo Parametric 2.0 are not supported.
  - the functions `ProXsecParallelCreate()` and `ProXsecPlanarWithoptionsCreate()` create cross sections as features.
  - the functions `ProXsecParallelCreate()` and `ProXsecPlanarWithoptionsCreate()` automatically convert the legacy cross sections to new cross section features as defined in Creo Parametric 2.0 before creating any new cross section feature.

The function `ProXsecOffsetCreate()` is deprecated. Use the function `ProXSectionOffsetCreate()` instead.

The function `ProXSectionOffsetCreate()` creates an offset cross section from a polyline. The polyline lies on a plane and the plane is defined by a local coordinate system. Offset cross section is created by extruding the polyline perpendicular to the sketching plane. The input arguments are:

- *solid_owner*—Specifies the model where the cross section will be created.
- *xsec_name*—Specifies the name of the cross section.
- *trf*—Specifies the local coordinate system of the plane which contains the polyline.
- *ent_arr*—Specifies a `ProArray` of `Pro2dEntdef` structure. The structure contains information about the entities of the polyline.
- *side*—Specifies the side to which the cross section must be extended. The cross section is extended normal to the polyline plane. The side is specified using the enumerated data type `ProXsecOffsetSide`. The valid values are:
  - `PRO_XSEC_OFFSET_BOTH_SIDES`—Extends the cross section to both sides of polyline plane.
  - `PRO_XSEC_OFFSET_SIDE_1`—Extends the cross section to the positive normal of polyline plane.
  - `PRO_XSEC_OFFSET_SIDE_2`—Extends the cross section to the negative normal of polyline plane.
- *flip*—Specifies the direction in which the cross section will be clipped. The value `False` indicates that the material on the right side of the polyline plane is retained. When the argument `side` is set to `PRO_XSEC_OFFSET_SIDE_1` or `PRO_XSEC_OFFSET_SIDE_2` then the material is retained from positive or negative side of polyline plane respectively.

When the value is set to `True` the above area is removed. The remaining material is retained.

Note

If the polyline from which the cross section has been created is closed, then flip works a little different.

Polyline Direction	Flip Value	Description
Closed polyline created clockwise	`False`	The material inside of the closed polyline is retained. When the argument `side` is set to `PRO_XSEC_OFFSET_SIDE_1` or `PRO_XSEC_OFFSET_SIDE_2` then the material is retained from positive or negative side of polyline plane respectively.
Closed polyline created clockwise	`True`	The material described in above case on page is removed. The remaining material is retained.
Closed polyline created counter clockwise	`True`	The material inside of the closed polyline is retained. When the argument `side` is set to `PRO_XSEC_OFFSET_SIDE_1` or `PRO_XSEC_OFFSET_SIDE_2` then the material is retained from positive or negative side of polyline plane respectively.
Closed polyline created counter clockwise	`False`	The material described in above case on page is removed. The remaining material is retained.

- *excld_incld_opt*—Specifies the items to be excluded from cutting by the cross section and is defined by the enumerated data type `ProXsecExcludeModels`. The items to be excluded are specified by the input parameter *exclude_items*.
- *exclude_items*—Specifies a `ProArray` of selected bodies or parts to include or exclude from the cross section.
- *data*—This option is specified using the data structure `ProXSectionCreateData` and is used to set quilt cross section type as `PRO_XSECTYPE_QUILTS` or `PRO_XSECTYPE_MODELQUILTS` for offset cross section. . This input argument is optional. If you do not want to offset cross section to cut quilts, set this as `Null`.

The functions `ProXSectionPlanarCreate()` and `ProXSectionOffsetCreate()` return an error `PRO_TK_LINE_TOO_LONG`, when the *xsec_name* is longer than `PRO_NAME_SIZE`.

The function `ProXSectionCreateDataAlloc()` allocates memory for the `ProXSectionCreateData` data structure.

Use the function `ProXSectionCreateDataFree()` to free the `ProXSectionCreateData` data structure memory.

The data structure `ProXSectionCreateData` is defined as follows:

Use the function `ProXSectionCreateDataQuiltSelGet()` to retrieve the quilt selection data. The output argument `r_quilt_sel` is the address of the quilt selection pointer given through the `ProSelection` object.

The function `ProXSectionCreateDataQuiltSelSet()` sets the quilt selection data using the structure `ProXSectionCreateData`.

The function `ProXSectionCreateDataQuiltTypeGet()` gets the quilt cross section type using the structure `ProXSectionCreateData`. The output argument `r_xsec_type` is defined by the enumerated data type `ProXsecCutobj`. The valid values can be `PRO_XSECTYPE_QUILTS` or `PRO_XSECTYPE_MODELQUILTS`.

Use the function `ProXSectionCreateDataQuiltTypeSet()` to set the quilt cross section type using the structure `ProXSectionCreateData`.

The function `ProXsecMakeVisible()` displays the specified cross section in the model. Use the function `ProXsecIsVisible()` to check if the specified cross section is displayed in the model.

The function `ProXsecActiveSet()` sets the specified cross section as active in the current view. Use the function `ProXsecActiveGet()` to retrieve the cross section, which is active in the current view.

Use the function `ProXsecCanCreateAsFeature()` to check if new cross section features can be created in the specified model. The function returns `PRO_B_FALSE` if the specified model has legacy cross sections.

The function `ProXsecOldToNewConvert()` converts the legacy cross sections to new cross section features as defined in Creo Parametric 2.0 for the specified model.

Use the function `ProXsecIsFeature()` to check whether the cross section is a feature.

The function `ProXsecFeatureGet()` returns a pointer to the cross section feature. The function returns the error type `PRO_TK_BAD_CONTEXT` for legacy cross sections.

The function `ProXsecDelete()` deletes a given cross section from a part or assembly.

Mass Properties of Cross Sections

Function Introduced:

- **ProXsecMassPropertyCompute()**

The function `ProXsecMassPropertyCompute()` calculates the mass properties of the cross section in the specified coordinate system. The function needs the name of a coordinate system datum whose X- and Y-axes are parallel to the cross section. The output from this function also refers to the coordinate system datum. Call `ProXsecRegenerate()` before `ProXsecMassPropertyCompute()`.

Note

The function `ProXsecMassPropertyCompute()` is not supported for offset and quilt type of cross sections.

Line Patterns of Cross Section Components

Functions Introduced:

- **ProXsecCompXhatchGet()**
- **ProXsecCompXhatchAdd()**
- **ProXsecCompXhatchReplace()**
- **ProXsectionCompXhatchStyleGet()**
- **ProXSectionItemXhatchStyleGet()**
- **ProXsecNewXhatchStyleCreateFromName()**

- **ProXsectionCompXhatchStyleSet()**
- **ProXSectionItemXhatchStyleSet()**

Superseded Functions:

- **ProXsecCompNewXhatchStyleSetByName()**
- **ProXsecCompNewXhatchStyleSet()**
- **ProXsecCompXhatchStyleSet()**
- **ProXsecCompXhatchStyleGet()**
- **ProXsecCompNewXhatchStyleGet()**

Creo Parametric supports hatch pattern files of the `*.pat` file format. The new hatch supports nonlinear hatching styles.

The old hatch uses the Xhatch `*.xch` file format. It is recommended to use the `*.pat` files. Refer to the Creo Parametric Online Help for more information.

The function `ProXsecCompXhatchGet()` returns the line patterns of a cross section component based on the specified cross section handle and the ID of the cross section component. The line patterns obtained are `ProXsecXhatch` structures that contain the following fields:

- `angle`—Specifies the angle of the line patterns.
- `spacing`—Specifies the distance between the line patterns.
- `offset`—Specifies the offset of the first line in the pattern.

Note

The functions `ProXsecCompXhatchGet()`, `ProXsecCompXhatchAdd()`, and `ProXsecCompXhatchReplace()` support only the old hatching styles, that is, the `*.xch` file format.

The function `ProXsecCompXhatchAdd()` adds a line pattern to a specified cross section component. This function takes the handle to the cross section, the ID of the cross section component, the handle to the drawing view containing the cross section component and a pointer to the `ProXsecXhatch` object as its input arguments.

Note

If the cross section component already includes a line pattern, then the function `ProXsecCompXhatchAdd()` does not add a line pattern.

The function ProXsecCompXhatchReplace() replaces all existing line patterns of a specified cross section component with a new one.

In Creo Parametric 7.0.0.0 the functions ProXsecCompXhatchStyleGet() and ProXsecCompNewXhatchStyleGet() are deprecated. Use the function ProXsectionCompXhatchStyleGet () to get p_xhatch_style from a component and ProXSectionItemXhatchStyleGet() to get p_xhatch_style from a body.

The function ProXSectionItemXhatchStyleGet() returns the cross section p_xhatch_style for the cross section handle of the body. The output argument p_xhatch_style returns a ProXsecNewXhatchStyle handle.

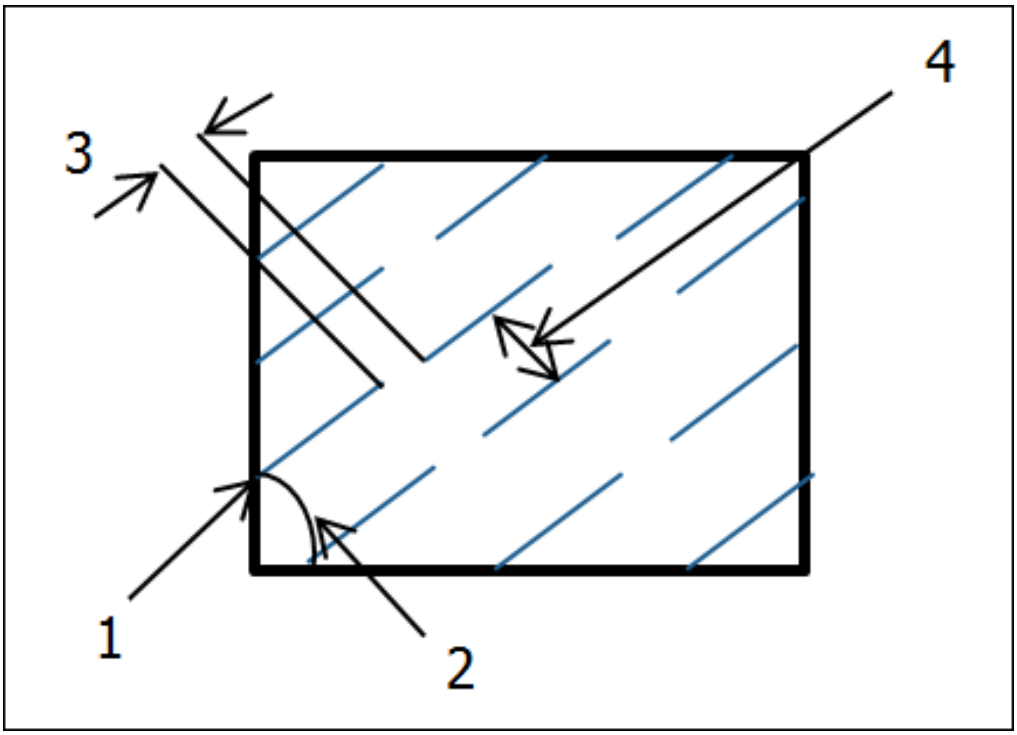
The function ProXsectionCompXhatchStyleGet() returns information about the style of hatch pattern in the specified cross section component. The output argument p_xhatch_style returns a ProXsecNewXhatchStyle handle. The structure ProXsecNewXhatchStyle specifies the following information:

- type—Type of hatch. Specify the following values:
  - PRO_XHATCH—Cross section component is appears as a hatch.
  - PRO_XSEC_EXCLUDED—Cross section is excluded for the specified assembly component.
  - PRO_XSEC_FILL—Cross section component appears as a solid.
  - PRO_XHATCH_ERASED—Cross section component is erased, that is, neither hatched nor filled.
- ProXsecXhatchPattern—Structure that contains information about the old hatch pattern in a cross section. It specifies the following information:
  - angle—Angle of the lines in the patterns.
  - spacing—Distance between the lines in the patterns.
  - offset—Offset of the first line in the pattern.
  - font—Line style for the line pattern.
  - color—Color for the line pattern.

Note

When the cross section has old hatch patterns, the field `*new_lines` in the structure `ProXsecNewXhatchStyle` is returned as `NULL`.

- `ProXsecNewXhatchPattern`—Structure that contains information about the new hatch pattern in a cross section. It specifies the following information:



  - 1 X-Origin, Y-Origin
  - 2 Angle
  - 3 Delta X
  - 4 Delta Y
  - `angle`—Angle of the lines in the patterns.
  - `x_origin`—Origin of the first pattern line along the x-axis.
  - `y_origin`—Origin of the first pattern line along the y-axis.
  - `delta_x`—For a dashed line pattern, the distance after which the next dashed line starts in a pattern.
  - `delta_y`—For a continuous and dashed line pattern, the distance between the pattern lines.
  - `dash`—Dashed line pattern.

- num_dash—Number of dashes for the line pattern. Max number of dashes is 6.
- color—Color for the line pattern.

Note

When the cross section has new hatch patterns, the field *old_lines in the structure ProXsecNewXhatchStyle is returned as NULL.

In Creo Parametric 7.0.0.0 the function ProXsecCompNewXhatchStyleSetByName() is deprecated. Use the function ProXsecNewXhatchStyleCreateFromName () to create ProXsecNewXhatchStyle. Use the functions ProXsectionCompXhatchStyleSet() and ProXSectionItemXhatchStyleSet() to set ProXsecNewXhatchStyle.

Use the function ProXsecNewXhatchStyleCreateFromName() to create a ProXsecNewXhatchStyle structure using the hatch pattern specified by the input argument *hatch_name*. This function supports only new, that is, PAT hatch patterns. The input arguments follows:

- *hatch_name*—Name of the existing PAT hatch.
- *color* —Color of the newly created pattern.
- *type*—Type of the hatch specified by structure ProXsecNewXhatchStyle and has the following values:
  - PRO_XHATCH—Cross section component appears as a hatch.
  - PRO_XSEC_EXCLUDED—Cross section is excluded for the specified assembly component.
  - PRO_XSEC_FILL—Cross section component appears as a solid.
  - PRO_XHATCH_ERASED—Cross section component is erased, that is, neither hatched nor filled.

After creating the ProXsecNewXhatchStyle structure using the function ProXsecNewXhatchStyleCreateFromName(), either of the following situations might occur:

- The new PAT hatch that is created using the function ProXSectionItemXhatchStyleSet() is available in the hatch edit dialog in the part and assembly mode and a suffix _TK is added to the hatch name.
- If the hatch name is used as the input parameter to the function ProXSectionItemXhatchStyleSet() is used, then a new hatch is not added.

The output argument `p_xhatch_style` can be set by the functions `ProXSectionItemXhatchStyleSet()` and `ProXsectionCompXhatchStyleSet()`.

The function `ProXSectionItemXhatchStyleSet()` sets the cross section `p_xhatch_style` for the cross section handle of the body, using the `ProXsecNewXhatchStyle` structure. The input arguments follow:

- *xsec_item*—Cross section handle of the specific body.
- *p_view*—View handle.
- *hatch_name*—Name of the nonlinear hatch.
- *p_xhatch_style*—Handle to `ProXsecNewXhatchStyle`. The unused hatch field must be set to `NULL`.

In Creo Parametric 7.0.0.0 the function `ProXsecCompXhatchStyleSet()` is deprecated. Use the function `ProXsectionCompXhatchStyleSet()` to set *p_xhatch_style* on a component and `ProXSectionItemXhatchStyleSet()` to set *p_xhatch_style* on a body.

Note

Use the function `ProXsecCompNewXhatchStyleGet()` for nonlinear hatch support.

The function `ProXsectionCompXhatchStyleSet()` sets the cross section `p_xhatch_style` for the specified component using the `ProXsecNewXhatchStyle` structure. The input arguments follow:

- *xsec*—Cross section handle of the specific body.
- *path*—Path to the specified component.
- *p_view*—View handle.
- *hatch_name*—Name of the nonlinear hatch.
- *p_xhatch_style*—Handle to `ProXsecNewXhatchStyle`. The unused hatch field must be set to `NULL`.

Note

`PRO_XSEC_EXCLUDED` type is applied only in the drawing environment.

When parts with multiple solid bodies are being cut by the cross section, the following functions return the error `PRO_TK_MULTIBODY_UNSUPPORTED`:

- `ProXsecCompNewXhatchStyleSetByName`
- `ProXsecCompNewXhatchStyleSet()`

- `ProXsecCompXhatchStyleSet()`
- `ProXsecCompNewXhatchStyleGet()`
- `ProXsecCompXhatchStyleGet()`

13

Core: Utilities

This chapter describes the Creo Parametric TOOLKIT utility functions.

Configuration Options

Functions Introduced:

- **ProConfigoptionGet()**
- **ProConfigoptSet()**
- **ProConfigoptArrayGet()**
- **ProDisplistInvalidate()**

The functions `ProConfigoptionGet()` and `ProConfigoptSet()` enable you to retrieve and set the current value for the specified configuration file option. The function `ProConfigoptGet()` has been deprecated. Use the function `ProConfigoptionGet()` instead. The function `ProConfigoptionGet()` returns the value of configuration option as a `ProPath` object.

To use `ProConfigoptSet()` on a configuration option that affects the display of Creo Parametric, you must call the function `ProDisplistInvalidate()` before you repaint the screen. This function makes sure Creo Parametric invalidates the two- or three-dimensional display list. The calling sequence of functions is as follows:

```
ProConfigoptSet (woption, value);
ProMdlCurrentGet (&model);
ProDisplistInvalidate (model)
ProWindowRepaint (-1);
```

The function `ProConfigoptSet()`, when applied to a multi string configuration option like "search_path", adds a new path entry into the session. It does not affect existing values. When applied to a single-valued configuration option, `ProConfigoptSet()` modifies the value of the configuration option.

The function `ProConfigoptArrayGet()` retrieves the value of a specified configuration file option. The function returns an array of values assigned to the configuration file. It returns a single value if the configuration file option is not a multi-valued option.

Registry File Data

Functions Introduced:

- **ProToolkitApplExecPathGet()**
- **ProToolkitApplTextPathGet()**

The function `ProToolkitApplExecPathGet()` returns the path to the Creo Parametric TOOLKIT executable file (`exec_file`) from the registry file.

The function `ProToolkitApplTextPathGet()` returns the path to the directory containing the `"text"` directory for the application.

Trail Files

Function Introduced:

- **ProTrailfileCommentWrite()**

To append a comment to the end of the current trail file, call the function `ProTrailfileCommentWrite()`. The comment should not be longer than (`PRO_COMMENT_SIZE` - 2) characters, and should not contain any nonprintable characters, such as “\n.”

Creo Parametric License Data

Function Introduced:

- **ProOptionOrderedVerify()**

The function `ProOptionOrderedVerify()` reports whether a specified Creo Parametric license option (such as Pro/MESH) is currently available in the Creo Parametric session.

Current Directory

Functions Introduced;

- **ProDirectoryCurrentGet()**
- **ProDirectoryChange()**

These two functions are concerned with the current default directory in Creo Parametric—the one in which it searches when you retrieve an object, for example. The Creo Parametric user changes this directory using the command **File ► Manage Session ► Working Directory**.

The function `ProDirectoryChange()` enables you to do the exact equivalent of **File ► Manage Session ► Working Directory** in Creo Parametric. Use this function if you need to save and retrieve objects in a directory other than the one the user chose.

The function `ProDirectoryCurrentGet()` returns the whole path to the directory, as a wide string.

File Handling

Functions Introduced:

- **ProFilesList()**
- **ProFileMdlnameOpen()**
- **ProFileMdlfiletypeOpen()**

- **ProFileOpenRegister()**
- **ProFileMdlnameSave()**
- **ProFileMdlfiletypeSave()**
- **ProFileSaveRegister()**
- **ProDirectoryChoose()**
- **ProFileMdlnameParse()**
- **ProPathMdlnameCreate()**
- **ProInfoWindowDisplay()**
- **ProFileEdit()**
- **ProTexturePathGet()**

The function `ProFilesList()` provides a list of the contents of a directory, given the directory path. You can filter the list to include only files of a particular type, as specified by the file extension. Use the `PRO_FILE_LIST_ALL` option to include all versions of a file in the list; use `PRO_FILE_LIST_LATEST` to include only the latest version. In addition to an array of file names, the function returns an array of subdirectory names, regardless of the filter used.

Starting with Pro/ENGINEER Wildfire 5.0, the function `ProFilesList()` can also list instance objects when accessing Windchill workspaces or folders. A PDM location (for workspace or commonspace) must be passed as the directory path. The following options have been added in the `ProFileListOpt` enumerated type that can be passed as the `listing_option` argument to `ProFilesList()`:

- `PRO_FILE_LIST_ALL_INST`—Same as the LIST_ALL option. It returns instances only for PDM locations.
- `PRO_FILE_LIST_LATEST_INST`—Same as the LIST_LATEST option. It returns instances only for PDM locations.
- `PRO_FILE_LIST_ALL_SORTED_INST`—Same as the `LIST_ALL_SORTED` option. It returns instances only for PDM locations.
- `PRO_FILE_LIST_LATEST_SORTED_INST`—Same as the `LIST_LATEST_SORTED` option. It returns instances only for PDM locations.
- `PRO_FILE_LIST_LATEST_SORTED_INST`—Same as the `LIST_LATEST_SORTED` option. It returns instances only for PDM locations.

The function `ProFileMdlnameOpen()` opens the dialog box to browse directories and open files. The function lets you specify the title of the dialog box, a set of shortcuts to other directories, and the name of a file to be preselected. This function uses the same filtering method as `ProFilesList()`. You can set a filter in the dialog box to include files of a particular type. In the input argument *filter_string* specify all types of files extensions with wildcards separated by commas, for example, `*.prt`, `*.asm`, `*.txt`, `*.avi`, and so on.

You can also use the function `ProFileMdlfiletypeOpen()` to browse directories and open files. You can set a filter in the dialog box to include files of a particular type. In the input argument `file_types`, you can specify an array of file types using the enumerated data type `ProMdlfileType`.

Note

The functions `ProFileMdlnameOpen()` and `ProFileMdlfiletypeOpen()` do not actually open the file, but return the file path of the selected file. For example, to open a text file, use the function `ProFileEdit()` or `ProInfoWindowDisplay()`.

The function `ProFileOpenRegister()` registers a new file type in the **File ▸ Open** dialog box in Creo Parametric. This function takes the access function `ProFileOpenAccessFunction()` and the action function `ProFileOpenOperationAction()` as its input arguments.

The function `ProFileOpenAccessFunction()` is called to determine whether the new file type can be opened using the **File ▸ Open** dialog box. The function `ProFileOpenOperationAction()` is called on clicking **Open** for the newly registered file type.

The function `ProFileMdlnameSave()` opens the save dialog box. This function has input arguments similar to `ProFileMdlnameOpen()`.

You can also use the function `ProFileMdlfiletypeSave()` to open the save dialog box. You can set a filter in the dialog box to include files of a particular type. In the input argument `file_types`, you can specify an array of file types using the enumerated data type `ProMdlfileType`.

Note

- The functions `ProFileMdlnameSave()` and `ProFileMdlfiletypeSave()` do not actually save the file, but return the file path of the selected file.
- For multi-CAD models, in a linked session of Creo Parametric with Windchill, the functions `ProFileMdlnameSave()` and `ProFileMdlfiletypeSave()` do not support a file path location on local disk.

The function `ProFileSaveRegister()` registers a new file type in the **File ▸ Save a Copy** dialog box in Creo Parametric. This function takes the access function `ProFileSaveAccessFunction()` and the action function `ProFileSaveOperationAction()` as its input arguments.

The function `ProFileSaveAccessFunction()` is called to determine whether the new file type can be saved using the **File ▶ Save a Copy** dialog box. The function `ProFileSaveOperationAction()` is called on clicking `OK` for the newly registered file type.

The function `ProDirectoryChoose()` prompts the user to select a directory using the Creo Parametric dialog box for browsing directories. Specify the title of the dialog box, a set of shortcuts to other directories, and the default directory path to start browsing. If the default path is specified as null, the current directory is used. The function returns the selected directory path as output.

In general, the file utility functions refer to files using a single wide character string, which contains four, distinct pieces of information that uniquely identify the file: the directory path, file name, extension, and version. The function `ProFileMdlnameParse()` takes such a string as input, and returns the four segments as separate arguments.

Note

The function `ProFileMdlnameParse()` returns the file version as `-1`, if the input argument *file_name_w_path* has the path specified as the path to a Windchill model. This function does not support fetching of model version of a Windchill model.

The function `ProPathMdlnameCreate()` performs the opposite operation—it builds the single wide string that identifies the file, given the path, file name, extension, and version.

The function `ProInfoWindowDisplay()` creates a text information window. It reads the contents from a text file in the current directory whose name is an input to the function. The function can also override the default size, shape, and location of the window. (These do not affect the properties of the Creo Parametric Information Window.)

The function `ProFileEdit()` opens an edit window on a specified text file. The editor used is the current default editor for Creo Parametric.

The function `ProTexturePathGet()` looks for the full path to the specified texture, decal, or bump map files and loads them from the texture path.

Note

For textures embedded inside a Creo Parametric model, if the `create_temp_file` is set to `true` the `ProTexturePathGet()` function writes a temporary copy of the specified files.

Wide Strings

Functions Introduced:

- **ProStringToWstring()**
- **ProWstringToString()**
- **ProWcharSizeVerify()**

These three utilities are described in the section Wide Strings on page 65 in the Fundamentals on page 22 chapter.

Freeing String Outputs

Many Creo Parametric TOOLKIT functions provide outputs of non-fixed length strings or wide strings. These outputs must be freed using a special set of functions, because they have been allocated by a special function internally.

Note

These functions must be only used for strings and string arrays output from Creo Parametric TOOLKIT functions. Check the function description to determine the function to use when freeing the output.

Functions Introduced:

- **ProStringFree()**
- **ProWstringFree()**
- **ProStringarrayFree()**
- **ProWstringarrayFree()**
- **ProStringproarrayFree()**
- **ProWstringproarrayFree()**

Use the functions `ProStringFree()` and `ProWstringFree()` to free a single `char*` or `wchar_t*` output from a Creo Parametric TOOLKIT function.

Use the functions `ProStringarrayFree()` and `ProWstringarrayFree()` to free a standard array of `char*` or `wchar_t*` output from a Creo Parametric TOOLKIT function.

Use the functions `ProStringproarrayFree()` and `ProWstringproarrayFree()` to free a `ProArray` of `char*` or `wchar_t*` output from a Creo Parametric TOOLKIT function.

Freeing Integer Outputs

Functions Introduced:

- **ProIntarrayFree()**

Use the function `ProIntarrayFree()` to free a plain integer array, which has been returned from a Creo Parametric TOOLKIT function.

Running Creo ModelCHECK

Creo ModelCHECK is an integrated application that runs transparently within Creo Parametric. Creo ModelCHECK uses a configurable list of company design standards and best modeling practices. You can configure Creo ModelCHECK to run interactively or automatically when you regenerate or save a model.

Functions Introduced:

- **ProModelcheckExecute()**

You can execute Creo ModelCHECK from an external application using the function `ProModelcheckExecute()`. The input parameters of this function are:

- `mdl`—Specifies the model on which you want to execute Creo ModelCHECK.
- `show_ui`—Specifies True to display the Creo ModelCHECK report in the Web browser.

> **Note**
>
> The configuration option `SHOW_REPORT` in the `config_init.mc` file overrides the `show_ui` value.

- `mcMode`—Specifies the mode in which you want to run Creo ModelCHECK. The modes are:
  - `PRO_MODELCHECK_GRAPHICS`—Interactive mode
  - `PRO_MODELCHECK_NO_GRAPHICS`—Batch mode
- `config_dir`—Specifies the location of the configuration files. If this parameter is set to NULL, the default Creo ModelCHECK configuration files are used.
- `output_dir`—Specifies a location for the reports. If this parameter is set to NULL, the default Creo ModelCHECK output directory is used.

The output parameters of this function are:

- errors—Specifies the number of errors found.
- warnings—Specifies the number of warnings found.
- model_saved—True if the model is saved with updates, else false.

Creating Custom Checks

This section describes how to define custom checks in Creo ModelCHECK that users can run using the standard Creo ModelCHECK interface in Creo Parametric.

To define and register a custom check:

1. Set the CUSTMTK_CHECKS_FILE configuration option in the start configuration file to a text file that stores the check definition. For example:

   CUSTMTK_CHECKS_FILE text/custmtk_checks.txt.
2. Set the contents of the CUSTMTK_CHECKS_FILE file to define the checks. Each check should list the following items:
   - DEF_<checkname>—Specifies the name of the check. The format must be CHKTK_<checkname>_<mode>, where mode is PRT, ASM, or DRW.
   - TAB_<checkname>—Specifies the tab (category) in the Creo ModelCHECK report under which the check is classified. Valid tab values are:
     - INFO
     - PARAMETER
     - LAYER
     - FEATURE
     - RELATION
     - DATUM
     - MISC
     - VDA
     - VIEWS
   - MSG_<checkname>—Specifies the description of the check that appears in the lower part of the Creo ModelCHECK report when you select the name.
   - DSC_<checkname>—Specifies the name of the check as it appears in the Creo ModelCHECK report table.
   - ERM_<checkname>—If set to INFO, the check is considered an INFO check and the report table will display the text from the first item returned

by the check, instead of a count of the items. Otherwise, this value must be included, but is ignored by Creo Parametric.

See Example 1: Text File for Custom Checks on page 273 for a sample custom checks text file.

1. Add the check and its values to the Creo ModelCHECK configuration file. For an example of how this is done, see the sample code at the end of this section.
2. Register the Creo ModelCHECK check from the Creo Parametric TOOLKIT application.

Note

Other than the requirements listed above, Creo Parametric TOOLKIT custom checks do not have access to the rest of the values in the Creo ModelCHECK configuration files. All the custom settings specific to the check such as start parameters, constants, and so on, must be supported by the user application and not Creo ModelCHECK. In the custom checks text file, separate options and values for options with a space, not by a tab character.

Functions Introduced

- **ProModelcheckCheckRegister()**
- **ProModelcheckCheckFunction()**
- **ProModelcheckUpdateFunction()**
- **ProModelcheckCleanupFunction()**

The function `ProModelcheckCheckRegister()` registers the callback functions for each custom check. The following arguments are available:

- The custom check name. This name must match the name given to the check in the Creo ModelCHECK configuration file, and must begin with "CHKTK_ ".
- The check label. Currently unused; the label is taken from the check configuration file.
- The check options. Currenlty unused, specify the value as `NULL`.
- The check callback functions, described in details below.
- Labels for the action and update buttons, or specify NULL if these buttons should not be shown.
- Application data to pass to the callback functions.

The check callback functions are as follows:

- The check function (required)—This function, whose signature matches `ProModelcheckCheckFunction()`, should calculate the results of the check and provide them to Creo ModelCHECK through the function output arguments.
- The action and update functions (optional)—These functions are called when users choose the action or update button when viewing the Creo ModelCHECK report. These have the signature of the `ProModelcheckUpdateFunction()` function.
- The cleanup function (optional)—Gives your application a chance to free memory allocated by the check function. This has the signature of the `ProModelcheckCleanupFunction()` function.

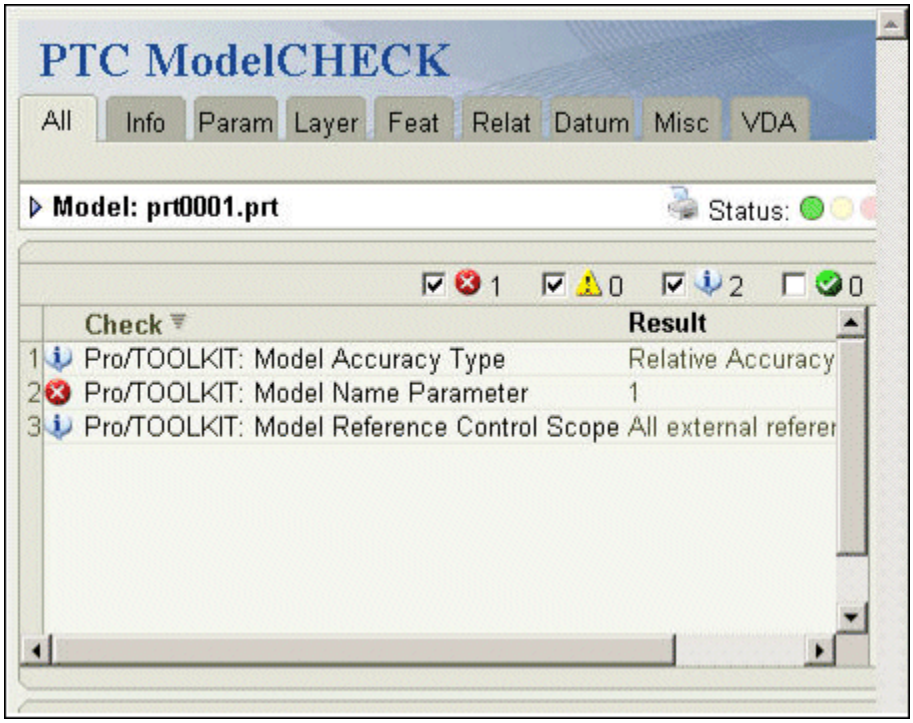
Each callback function receives the following inputs:

- The name of the custom check, as defined in the original call to `ProModelcheckCheckRegister()`.
- The model being checked.
- A pointer to the application data, set during the call to `ProModelcheckCheckRegister()`.

The function whose prototype matches `ProModelcheckCheckFunction()` is used to evaluate the custom defined checks. The user application runs the check on the specified model and populates the following output arguments:

- *results_count*—Specifies an integer indicating the number of errors found by the check.
- *results_url*—Specifies the link to an application-owned page that contains information specific to the check.
- *results_table*—Specifies an array of data for each error encountered that will be shown along with the results.

The following figure illustrates how the results of some custom checks might be displayed in the Creo ModelCHECK report.



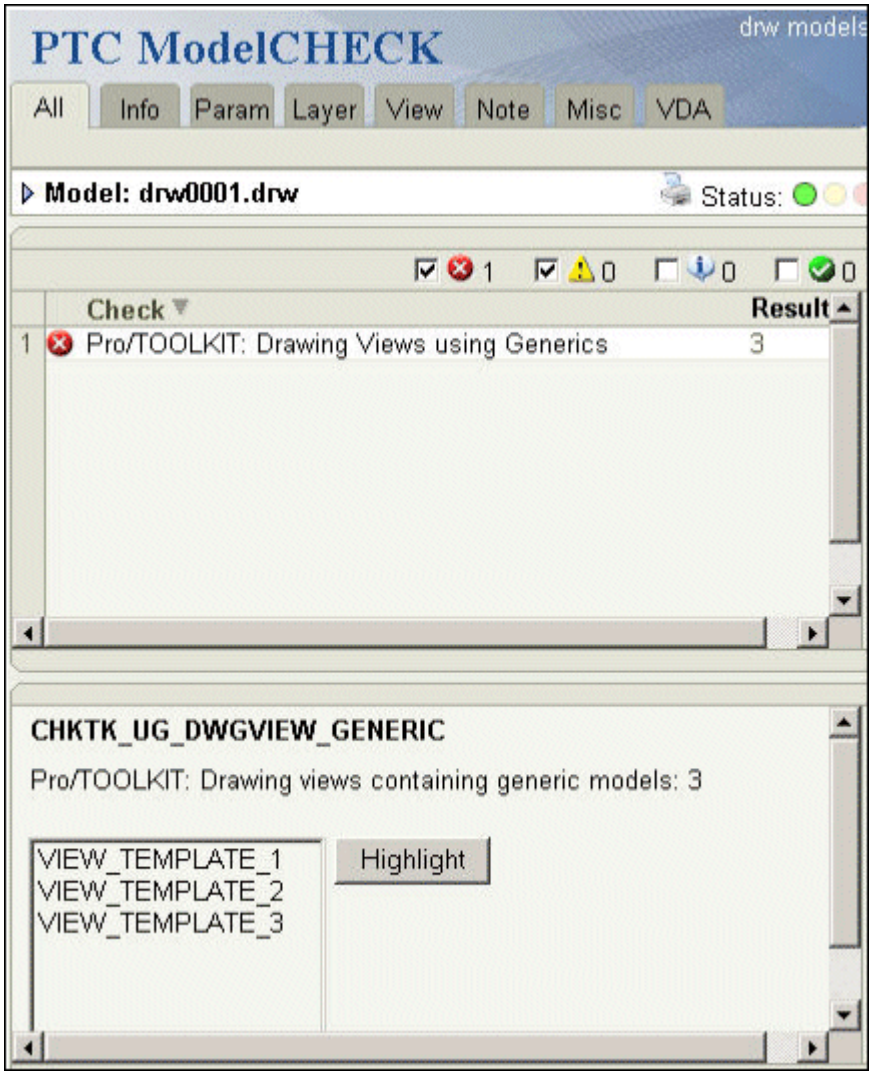
The function whose prototype matches `ProModelcheckCleanupFunction()` is used to release memory allocated for the check callback function.

The functions whose prototypes matches `ProModelcheckUpdateFunction()` are used for the following:

- To execute a check-defined action on a given item.
- To automatically update a given item to fix errors, if any.

The selected item's description string is passed as an argument to the update function.

Creo ModelCHECK checks may have one "action" function to highlight the problem, and possibly an update function, to fix it automatically. The following figure displays the Creo ModelCHECK report with an action button that invokes the action callback function.



Example 1: Text File for Custom Checks

The following is the text file for the custom check examples.

```
UG CustomCheck: MDL PARAM NOT FOUND
Parameter %0w is not found in the model %1w
#
#
UG CustomCheck: MDL PARAM OK
Parameter %0w is set correctly in the model %1w
#
#
UG CustomCheck: MDL PARAM INV TYPE
Parameter %0w in %1w is not a String parameter
#
#
Parameter %0w value does not match model name %1w
```

```
                UG CustomCheck: MDL PARAM INCORRECT
#
#
UG CustomCheck: MDL PARAM NAME
Parameter %0w value matches model name
#
#
UG CustomCheckUpdate: MDL PARAM NAME
Fix Parameter
#
#
%CIUG CustomCheck: MDL PARAM UPDATED
Mdl name parameter %0w has been updated.
#
#
%CEUG CustomCheck: MDL PARAM UPDATE TYPE
Cannot modify the type of parameter %0w; this error should be fixed
manually.
#
#
UG CustomCheck: MDL ACC TYPE
Model accuracy
#
#
UG CustomCheck: MDL ACC ABS
Absolute Accuracy
#
#
UG CustomCheck: MDL ACC REL
Relative Accuracy
#
#
UG CustomCheck: DWGVIEW GENERIC
Drawing Views using Generics
#
#
UG CustomCheckAction: DWGVIEW GENERIC
Highlight
#
#
```

Example 2: Registering Custom Creo ModelCheck Checks

The sample code in `UgModelCheck.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main` shows how to register custom Creo ModelCHECK checks using Creo Parametric TOOLKIT. The following custom checks are registered:

- CHKTK_MDL_NAME_PARAM—Determines if the model has a parameter whose value is equal to the model name
- CHKTK_MDL_REFC_SCOPE—Info check that prints the model reference control settings
- CHKTK_DWVIEW_GENERIC—Drawing mode check that looks for views that use generic models

Example 3: Implementing a Model Name Parameter Check

The sample code in UgModelCheck.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main shows the implementation for the model name parameter check. This check has a check function callback and an update function callback. Also included is the cleanup callback used for all of the custom checks.

Example 4: Implementing a Reference Control Info Check

The sample code UgModelCheck.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main shows the implementation for an info check that will report on the reference control setting applied to the model. This check has a check function callback but no action or update function (because it is an info-only check).

Example 5: Implementing a Check Looking for Drawing Views Using Generics

The sample code in UgModelCheck.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main shows the implementation for the drawing view using generics check. This check has a check function callback and an action function used to highlight the views that have this error.

Example 6: Changes to the Creo ModelCheck Configuration Files to enable Custom Checks

The following show the changes made to the Creo ModelCHECK configuration files to enable the custom checks.

```
Lines from the custom TK checks file (custmtk_checks.txt)
# Custom TK Check File
#
# UG_MDLPARAM_NAME
DEF_UG_MDLPARAM_NAME CHKTK_UG_MDLPARAM_NAME_PRT
TAB_UG_MDLPARAM_NAME PARAMETER
MSG_UG_MDLPARAM_NAME Pro/TOOLKIT: Models with missing or invalid model
name parameters:
ERM_UG_MDLPARAM_NAME  N/A
```

```
DSC_UG_MDLPARAM_NAME Pro/TOOLKIT: Model Name Parameter
# UG_MDL_REFC_SCOPE
DEF_UG_MDL_REFC_SCOPE
CHKTK_UG_MDL_REFC_SCOPE_PRT
TAB_UG_MDL_REFC_SCOPE INFO
MSG_UG_MDL_REFC_SCOPE Pro/TOOLKIT: Model reference control scope:
ERM_UG_MDL_REFC_SCOPE INFO
DSC_UG_MDL_REFC_SCOPE Pro/TOOLKIT: Model reference control scope:
# UG_DWGVIEW_GENERIC
DEF_UG_DWGVIEW_GENERIC CHKTK_UG_DWGVIEW_GENERIC_DRW
TAB_UG_DWGVIEW_GENERIC VIEWS
MSG_UG_DWGVIEW_GENERIC Pro/TOOLKIT: Drawing views containing generic
models:
ERM_UG_DWGVIEW_GENERIC N/A
DSC_UG_DWGVIEW_GENERIC Pro/TOOLKIT: Drawing Views using Generics
Lines added to the ModelCheck configuration file (default_checks.mch)
CHKTK_UG_MDLPARAM_NAME_PRT      YNEW        E  E  E  E  Y
CHKTK_UG_MDL_REFC_SCOPE_PRT     YNEW        Y  Y  Y  Y  Y
CHKTK_UG_DWGVIEW_GENERIC_DRW    YNEW        E  E  E  E  Y
Lines added to the ModelCheck start file (nostart.mcs)
CUSTMTK_CHECKS_FILE    text/custmtk_checks.txt
```

14

Core: Asynchronous Mode

This chapter explains using Creo Parametric TOOLKIT in Asynchronous Mode.

Overview

Asynchronous mode is a multiprocess mode in which the Creo Parametric TOOLKIT application and Creo Parametric can perform concurrent operations. Unlike synchronous mode, the asynchronous mode uses remote procedure calls (rpc) as the means of communication between the application and Creo Parametric.

Another important difference between synchronous and asynchronous modes is in the startup of the Creo Parametric TOOLKIT application. In synchronous mode, the application is started by Creo Parametric, based on information contained in the registry file. In asynchronous mode, the application is started independently of Creo Parametric and subsequently either starts or connects to a Creo Parametric process. The application can contain its own `main()` or `wmain()` function. Use `wmain()` if the application needs to receive command-line arguments as `wchar_t`, for example if the input contains non-usascii characters. Note that an asynchronous application will not appear in the Auxiliary Applications dialog box.

The section How Creo Parametric TOOLKIT Works on page 35 in Fundamentals on page 22 chapter describes two modes—DLL and multiprocess (or “spawned”). These modes are synchronous modes in the sense that the Creo Parametric TOOLKIT application and Creo Parametric do not perform operations concurrently. In spawn mode, each process can send a message to the other to ask for some operation, but each waits for a returning message that reports that the operation is complete. Control alternates between the two processes, one of which is always in a wait state.

Asynchronous mode applications operate with the same method of communication as spawn mode (multiprocess). The use of rpc in spawn mode causes this mode to perform significantly slower than DLL communications. For this reason, you should be careful not to apply asynchronous mode when it is not needed. Note that asynchronous mode is not the only mode in which your application can have explicit control over Creo Parametric. You can also run Creo Parametric in batch mode using Creo Parametric TOOLKIT applications; for more information on batch mode operation, refer to the section Using Creo Parametric TOOLKIT to Make a Batch Creo Parametric Session on page 52.

Setting up an Asynchronous Creo Parametric TOOLKIT Application

For your asynchronous application to communicate with Creo Parametric, you must set the environment variable `PRO_COMM_MSG_EXE` to the full path of the executable `pro_comm_msg.exe` , that is, `<creo_loadpoint>\<datecode>\Common Files\<machine>\obj\pro_comm_msg.exe`

Set `PRO_COMM_MSG_EXE` in the **Environment Variables** section of the **System** window (which can be accessed from the Control Panel).

Depending on how your asynchronous application handles messages from Creo Parametric, your application can be classified as either “simple” or “full”. The following sections describe simple and full asynchronous mode.

The environment variable `RPC_TIMEOUT` sets the maximum time limit in which the remote procedure calls must be executed. The variable sets the time in seconds. The default value is set to 2000 seconds. The functions return the error `PRO_TK_COMM_ERROR`, when the application times out without complete execution. If your application requires more time for execution, you must set this variable to a higher value.

Note

For asynchronous mode applications, you are not required to supply `user_initialize()` and `user_terminate()`, Creo Parametric TOOLKIT will supply a default implementation for these functions. If you do wish to include a custom version of either function, both the functions `user_initialize()` and `user_terminate()` must be supplied in the application. A custom `user_initialize()` will make it possible to access the `user_initialize()` input arguments and to reuse code from a synchronous application without making modifications.

Simple Asynchronous Mode

A simple asynchronous application does not implement a way to handle requests from Creo Parametric. Therefore, Creo Parametric cannot call functions in the Creo Parametric TOOLKIT application. Consequently, Creo Parametric cannot invoke the callback functions that must be supplied when you add, for example, menu buttons or notifications to Creo Parametric.

Despite this limitation, Creo Parametric running with graphics is still an interactive process available to the user.

When you design a Creo Parametric TOOLKIT application to run in simple asynchronous mode, keep the following in mind:

- The Creo Parametric process and the application perform operations concurrently.
- None of the application’s functions are invoked by Creo Parametric.
- Simple asynchronous mode supports Creo Parametric TOOLKIT visit functions (`ProSolidFeatVisit()`, for example), but does not support notification callbacks.

These considerations imply that the Creo Parametric TOOLKIT application does not know the state (the current mode, for example) of the Creo Parametric process at any moment.

Starting and Stopping Creo Parametric

Functions introduced:

- **ProEngineerStart()**
- **ProEngineerConnectionStart()**
- **ProEngineerEnd()**

A simple asynchronous application can spawn and connect to a Creo Parametric process via the function `ProEngineerStart()`. During this startup, Creo Parametric calls `user_initialize()` if it is present in the application. The Creo Parametric process "listens" for requests from the application and acts on the requests at suitable breakpoints—normally between commands.

The function `ProEngineerConnectionStart()` performs the same task as `ProEngineerStart()`, except that `ProEngineerConnectionStart()` outputs a ProProcessHandle which can be used for later connect and disconnect operations. Using this function requires building with a C++ compiler — see the description of `ProEngineerConnect()` for more information.

To connect to an existing Creo Parametric process from an asynchronous application, see the section Connecting to a Creo Parametric Process on page 280.

Unlike applications running in synchronous mode, asynchronous applications are not terminated when Creo Parametric terminates. This functionality is useful when the application needs to perform Creo Parametric operations only intermittently, and therefore start and stop Creo Parametric more than once during a session. To end a Creo Parametric process, call the function `ProEngineerEnd()`.

Connecting to a Creo Parametric Process

Functions introduced:

- **ProEngineerConnectIdGet()**
- **ProEngineerConnect()**
- **ProEngineerDisconnect()**
- **ProEngineerConnectIdExtract()**

The function `ProEngineerConnectIdGet()` returns the asynchronous connection id of the Creo Parametric session that the application is running with. This function can be called from any synchronous (DLL or spawn mode) or

asynchronous application. It allows the application to send the connection id to any other asynchronous application that needs to connect to this specific Creo Parametric session.

A simple asynchronous application can also connect to a Creo Parametric process that is already running on that machine. The function `ProEngineerConnect()` performs this function. It allows you to specify the name of the user who owns the Creo Parametric process to which you want to connect, and the name of the machine used for the display. If multiple Creo Parametric sessions meet this specification, `ProEngineerConnect()` can optionally choose one process at random or return an error status.

To disconnect from a Creo Parametric process, call `ProEngineerDisconnect()`.

The connection to a Creo Parametric process uses information that is provided by the name service daemon. The name service daemon accepts and supplies information about the processes running on the specified hosts. The application manager, for example, uses name service when it starts up Creo Parametric and other processes. The name service daemon is set up as part of the Creo Parametric installation.

The function `ProEngineerConnectIdExtract()` returns a string connection identifier for the Creo Parametric process. This identifier can be used later to connect to the same process using a call to `ProEngineerConnect()`. Pass the connection id as the first argument to the connection function.

To use the functions in this section, and also the function `ProEngineerConnectionStart()`, you must link your application with the library `pt_asynchronous.lib`, which is in the following location:
`<creo_toolkit_loadpoint>/<Machine>/obj`

Because this is a C++ library, you must use a C++ compiler to build your application. For sample makefiles containing C++ settings, see the makefiles under the directory `<creo_toolkit_loadpoint>/<Machine>/obj`

Note

You do not have to link with pt_asynchronous.lib (or use a C++ compiler) if you do not use the functions just described or `ProEngineerConnectionStart()`.

Status of a Creo Parametric Process

Function introduced:

- **ProEngineerStatusGet()**

It might be useful for your application to know whether a Creo Parametric process is running. The function `ProEngineerStatusGet()` returns this information.

Full Asynchronous Mode

Functions introduced:

- **ProEventProcess()**
- **ProAsynchronousEventLoop()**
- **ProAsynchronousEventLoopInterrupt()**
- **ProTermFuncSet()**
- **ProTerminationAction()**

Full asynchronous mode is identical to simple asynchronous mode except in the way the Creo Parametric TOOLKIT application handles requests from Creo Parametric. In simple asynchronous mode, it is not possible to process such requests. In full asynchronous mode, the application must implement a control loop that "listens" for messages that arrive from Creo Parametric. As a result, Creo Parametric can call functions in the application, including callback functions for menu buttons and notifications.

The control loop of an application running in full asynchronous mode must contain a call to the function `ProEventProcess()`, which takes no arguments. This function responds to Creo Parametric messages in a manner similar to synchronous mode. For example, if the user selects a menu button that is added by your application, `ProEventProcess()` processes the call to your callback function and returns when the call completes. (For more information on callback functions and adding menu buttons, see the User Interface: Menus, Commands, and Popupmenus on page 301 chapter.)

The function `ProAsynchronousEventLoop()` provides an alternative to the development of an event processing loop in a full asynchronous mode application. Call this function to have the application wait in a loop for events to be passed from Creo Parametric. No other processing will take place while the application is waiting. The loop will continue until `ProAsynchronousEventLoopInterrupt()` is called from a Creo Parametric TOOLKIT callback action, or until the application detects that Creo Parametric has terminated.

It is often necessary for your full asynchronous application to be notified of the termination of the Creo Parametric process. In particular, your control loop need not continue to listen for Creo Parametric messages if Creo Parametric is no longer running. The function `ProTermFuncSet()` binds a termination action to be executed when Creo Parametric is terminated. The termination action is a

function that you supply and identify in the input of `ProTermFuncSet()` by a function pointer of type `ProTerminationAction`. The input to the termination action is the termination type, which is one of the following:

- `PROTERM_EXIT`—Normal exit (the user picks **Exit** from the menu).
- `PROTERM_ABNORMAL`—Exit with error status.
- `PROTERM_SIGNAL`—Fatal signal raised.

Your application can interpret the termination type and take appropriate action.

Setting Up a Non-Interactive Session

You can spawn a Creo Parametric session that is both noninteractive and nongraphical. In asynchronous mode, include the following arguments in the call to `ProEngineerStart()`:

- *-g:no_graphics*—Turn off the graphics display.
- *-i:rpc_input*—Cause Creo Parametric to expect input from your asynchronous application only.

Both of these arguments are required, but the order is not important. The syntax of the call for a noninteractive, nongraphical session is as follows:

```
ProEngineerStart ("parametric -g:no_graphics -i:rpc_input", <text_dir>);
```

In the syntax, `parametric` is the command to start Creo Parametric.

15

User Interface: Messages

This chapter describes the functions used to communicate with the user through the text message area, including keyboard entry.

Writing a Message Using a Popup Dialog

The function `ProUIMessageDialogDisplay()` displays the UI message dialog. The input arguments to the function are:

- The type of message to be displayed.
- The text to display as the title of the dialog. If you want to support displaying localized text, use the message files and the function `ProMessageToBuffer()` to generate this string. Message files are described later in this chapter.
- The message text to be displayed in the dialog. If you want to support displaying localized text, use the message files and function `ProMessageToBuffer()` to generate this string. Message files are described later in this chapter.
- ProArray of possible button identifiers for the dialog.
- The identifier of the default button for the dialog.

The function outputs the button that the user selected.

Example 1: Displaying the UI Message Dialog

The sample code in `UgUIMessageDisplay` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_message` shows how to display a hard coded confirmation message.

Writing a Message to the Message Window

This section describes the following topics:

- Displaying and clearing messages
- The text message file

Functions Introduced:

- **ProMessageDisplay()**
- **ProMessageClear()**
- **ProUIMessageDialogDisplay()**

The function `ProMessageDisplay()` is similar to the C function `printf()`, but with some important differences:

- The first argument is the name (as a wide string) of the message file. The name must include the file extension, but not the path. See the section Text Message File Format and Restrictions on page 286.
- The second argument, instead of being a format string, is a keyword used to look up the format string in the message file.
- The subsequent arguments for the values inserted into the format string are pointers, not values. These values can be data inserted into the message or default values for the data to be read from user input. See the section Getting Keyboard Input on page 290 for more information.
- Although the list of arguments for the values is variable in number, there is a maximum of 9. See Contents of the Message File on page 287 for more information on using these arguments with a message format.

The function `ProMessageClear()` scrolls the text in the message area up one line. This could be used to indicate that Creo Parametric has received the user's response to a message.

Text Message File Format and Restrictions

The text message file enables you to provide your own translation of the text message, just as the menu files enable you to provide your own translation of the button name and the one-line command help.

Restrictions on the Text Message File

You must observe the following restrictions when you name your message file:

- The name of the file must be 30 characters or less, including the extension.
- The name of the file must contain lowercase characters only.
- The file extension must be three characters.
- The version number must be in the range 1 to 9999.
- All message file names must be unique, and all message key strings must be unique across all applications that run with Creo Parametric. Duplicate message file names or message key strings can cause Creo Parametric to exhibit unexpected behavior. To avoid conflicts with the names of Creo Parametric or Creo Parametric TOOLKIT application message files or message key strings, PTC recommends that you choose a prefix unique to your application, and prepend that prefix to each message file name and each message key string corresponding to that application.

Creo Parametric looks for the message file using the following search path:

- The current Creo Parametric directory
- The directory text under the directory named in the `text_dir` statement in the registry file (`protk.dat`).

Note that message files are loaded into Creo Parametric only once during a session, during the first call to `ProMessageDisplay()`. Consequently, if you make a change to the message file while Creo Parametric is running, you must exit and restart Creo Parametric to have the changes take effect.

Contents of the Message File

The message file consists of groups of four lines—one group for each message you want to write. The four lines are as follows:

1. A string that acts as the keyword to identify the message when you call `ProMessageDisplay()` This keyword must be unique for all Creo Parametric messages.
2. A string that will be substituted for the first string when you call `ProMessageDisplay()`. This string acts like the format string in a `printf()` statement. By modifying this line in the message file, you can modify the text of the message without modifying your C code.
3. The translation of the message into another language (can be blank).
4. An intentionally blank line reserved for future extensions.

The format string (line 2 in the message file) differs from the format string of a `printf()` in the following respects:

- The conversion specifications (`%d`, `%s`, and so on) must include an argument number corresponding to the position of that argument in the subsequent list (starting at 0). For example, instead of `%d`, `%s`, you must have `%0d`,`%1s`, and so on. If you want to specify a field width, put it in parentheses between the position number and the type specifier; for example, `%0(5.3)f`.
- The separator ||| between message text and a conversion specification signifies that the conversion specification is for a default value for user input. This default value will appear in the text box created using the keyboard input functions, such as `ProMessageIntegerRead()`. Refer to Using Default Values on page 290 for more on default values.
- The conversion character `w` is available for wide strings.
- You do not need the character constant (`\n`) at the end of the format. Creo Parametric automatically inserts a new line when necessary.

Note

The length of any line in the message file must not exceed 4096 wide characters.

The following table lists the conversion characters and their corresponding data types.

Conversion Character	Data Type
`f`	Float (or double)
`d`	Decimal integer
`s`	Ordinary string (or type `char[]`)
`w`	Wide character strings
`e`	Exponential
`g`	Either float or exponential, as appropriate

Ensure that the keyword string is similar to the format string to make your C code easy to interpret. Add a prefix that is unique to your application to the keyword string. The examples in this manual use the unique prefix "USER."

Message Classification

Messages displayed in Creo Parametric include a symbol which identifies the message type. Each message type is identified by a classification which begins with the characters `%C`. A message classification requires that the message key (line 1 in the message file) be preceeded by the classification code. Note that the message key string used in the code should NOT contain the classification.

Creo Parametric TOOLKIT applications can now display any or all of these message symbols:

- Prompt—the Creo Parametric message displayed is preceded by a green arrow. The user must respond to this message type (to either input information, accept the default value offered, or cancel the application). Without such action, no progress can be made. The response may be either textual or in the form of a selection. The classification code for prompt messages is `%CP`.
- Info—the Creo Parametric message displayed is preceded by a blue dot. This message type contains information such as user requests or feedback from either Creo Parametric or the Creo Parametric TOOLKIT application. The classification code for prompt messages is `%CI`.

Note

Do not classify as Info any message which informs users of a problem with an operation or process. These messages should be classified as Warnings.

- Warning—the Creo Parametric message displayed is preceded by a triangle containing an exclamation point. Warnings alert the user to situations which may lead to potentially erroneous situations at a later stage, for example, possible process restrictions imposed or a suspected data problem. However, warnings do not prevent or interrupt task completion, nor should they be used to indicate a failed operation. Warnings only caution the user that the operation has been completed, but may not have been performed in a completely desirable way. The classification code for prompt messages is `%CW`.
- Error—the Creo Parametric message is preceded by a broken square. This message type informs the user when a required task was not successfully completed. It may or may not require intervention or correction before work can continue, depending on the application. Whenever possible, provide a path to redress this situation. The classification code for prompt messages is `%CE`.
- Critical—the Creo Parametric message displayed is preceded by a red X. This message type informs the user of extremely serious situations, especially those which could cause the loss of user data. Options for redressing the situation (if available) should be provided with the message. The classification code for prompt messages is `%CC`.

Writing a Message to an Internal Buffer

Function Introduced:

- **ProMessageToBuffer()**
- **ProMessageWstringbufferAlloc()**

The functions `ProMessageToBuffer()` and `ProMessageWstringbufferAlloc()` have the same relationship to `ProMessageDisplay()` that the C library function `sprintf()` has to `printf()`: it enables you to write a message to an internal, wide-string buffer instead of to the Creo Parametric message area. These functions perform exactly the same argument substitution and translation as `ProMessageDisplay()`. You provide a wide-string buffer as the first argument, and the subsequent arguments are the same as those for `ProMessageDisplay()`.

The function `ProMessageToBuffer()` allows you to write a message with a maximum length of 80 characters to a wide-string buffer.

For a message of any length use the function `ProMessageWstringbufferAlloc()`. You must free the output string `translated_msg` using the function `ProWstringFree()`.

Getting Keyboard Input

Functions Introduced:

- **ProMessageIntegerRead()**
- **ProMessageDoubleRead()**
- **ProMessageStringRead()**
- **ProMessagePasswordRead()**

These four functions obtain keyboard input from a text box at the bottom of the Creo Parametric window. The functions check the syntax of the user's entry and indicate when the entry is simply a carriage return. Each of the functions enable you to restrict numeric input to a specified range, or string input to a specified string length. The functions continue to prompt the user until a valid response is entered.

The function `ProMessageStringRead()` supports string lengths up to 127 characters.

Using Default Values

Prior to Release 20, Pro/TOOLKIT applications implemented default values by checking for a user-entered carriage return. Beginning with Release 20, you can specify default values within the call to `ProMessageDisplay()`, provided that the separator ||| appears in the format string in the message file. (See the section Contents of the Message File on page 287 for the specific placement of the ||| separator.)

Note

You must call the function `ProMessageDisplay()` before calling the `ProMessage*Read` functions to specify the default values.

Default values are displayed in the text box as input. Note that this value will not be passed to the Creo Parametric TOOLKIT function if the user hits a carriage return; instead, the function will return `PRO_TK_GENERAL_ERROR` and the application must interpret that the user intends to use the default.

To specify a constant default value, the format string would appear as follows:

```
Enter a double: |||3.0
```

Specifying constant defaults is not recommended as changing the default requires revising the Creo Parametric TOOLKIT application. Specifying defaults that are variables is more flexible.

To specify a default integer that is a variable, for example, the format string in the message file would appear as follows:

```
Enter any integer: |||%0d
```

Example 2: Displaying Messages and Retrieving Keyboard Input

The sample code in `UgMessageWindowUse.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_message` shows how to print messages and accept keyboard input with default values. The example also shows how to write a message to an internal, wide-string buffer. The name of the message file is `msg_ugmessage.txt`.

16

User Interface: Ribbon Tabs, Groups, and Menu Items

This chapter describes theCreo Parametric TOOLKIT support for the Ribbon User Interface (UI). It also describes the impact of the ribbon user interface on legacy Pro/TOOLKIT applications and the procedure to place the commands, buttons, and menu items created by the legacy applications in the Creo Parametric ribbon user interface. Refer to the Creo Parametric Help for more information on the ribbon user interface and the procedure to customize the ribbon.

Creating Ribbon Tabs, Groups, and Menu Items

Customizations to the ribbon user interface using the Creo Parametric TOOLKIT applications are supported through the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. You can specify the user interface layout for a Creo Parametric TOOLKIT application and save the layout definition in a ribbon definition file, `toolkitribbonui.rbn`. When you run Creo Parametric, the `toolkitribbonui.rbn` file is loaded along with the Creo Parametric TOOLKIT application and the commands created by the Creo Parametric TOOLKIT application appear in the ribbon user interface. Refer to the section About the Ribbon Definition File on page 295for more information on the `toolkitribbonui.rbn` file.

You can customize the ribbon user interface only for a particular mode in Creo Parametric. For example, if you customize the ribbon user interface and save it to the `toolkitribbonui.rbn` file in the Part mode, then on loading Creo Parametric the customized user interface will be visible only in the Part mode. To view a particular tab or group in all the modes, you must customize the ribbon user interface and save the `toolkitribbonui.rbn` file in each mode. Refer to the Creo Parametric help for more information on customizing the ribbon.

Note

You can add a new group to an existing tab or create a new tab using the **Customize Ribbon** tab in the**Creo Parametric Options** dialog box. You will not be able to modify the tabs or groups that are defined by Creo Parametric.

Workflow to Add Menu Items to the Ribbon User Interface

Note

The instructions explained below are applicable only if the application is implemented in full asynchronous mode. This is because applications in simple asynchronous mode cannot handle requests, that is, command callbacks, from Creo Parametric. Refer to the chapter Core: Asynchronous Mode on page 277, for more information.

You must have a Creo Parametric TOOLKIT development license to customize the ribbon user interface for Creo Parametric TOOLKIT applications. The steps to add commands to the Creo Parametric ribbon user interface are as follows:

1. Create a Creo Parametric TOOLKIT application with complete command definition, which includes specifying command label, help text, large icon name, and small icon name. Designate the command using the `ProCmdDesignate()`.

   Note

   The labels and the text added using the `ProCmdDesignate()` function duplicate existing messages that are previously added in the Creo database. To display the correct label and text message, you can use a prefix or a suffix with the message names that will identify your Creo Parametric TOOLKIT application. You should avoid using generic names of Creo Parametric TOOLKIT buttons such as Point, Arc, Circle, Ellipse in the labels and text.

2. Start the Creo Parametric TOOLKIT application and ensure that it is started or connected to Creo Parametric. The commands created by the Creo Parametric TOOLKIT application will be loaded in Creo Parametric.
3. Click **File ► Options**. The **Creo Parametric Options** dialog box opens.
4. Click **Customize Ribbon**.
5. In the **Customize the Ribbon** list, select a tab and create a new group in it or create a new tab and a group in it.
6. In the **Choose commands from** list, select **TOOLKIT Commands**. The commands created by the Creo Parametric TOOLKIT application are displayed.
7. Click **Add** to add the commands to the new tab or group.
8. Click **Import/Export ► Save the Auxiliary Application User Interface**. The changes are saved to the `toolkitribbonui.rbn` file. The `toolkitribbonui.rbn` file is saved in the text folder specified in the Creo Parametric TOOLKIT application registry file. For more information refer to the section on Ribbon Definition File on page 295.
9. Click **Apply**. The custom settings are saved to the `toolkitribbonui.rbn` file.
10. Reload the Creo Parametric TOOLKIT application or restart Creo Parametric. The `toolkitribbonui.rbn` file will be loaded along with the Creo Parametric TOOLKIT application.

If translated messages are available for the newly added tabs or groups, then Creo Parametric displays the translated strings by searching for the same string from the list of string based messages that are loaded. For more information refer to the section on Localizing the Ribbon User Interface Created by the Creo Parametric TOOLKIT Applications on page 298.

About the Ribbon Definition File

A ribbon definition file is a file that is created through the **Customize Ribbon** interface in Creo Parametric. This file defines the containers, that is, Tabs, Groups, or Cascade menus that are created by a particular Creo Parametric TOOLKIT application. It contains information on whether to show an icon or label. It also contains the size of the icon to be used, that is, a large icon (32X32) or a small icon (16x16).

The ribbon user interface displays the commands referenced in the ribbon definition file only if the commands are loaded and are visible in that particular Creo Parametric mode. If translated messages are available for the newly added tabs or groups, then Creo Parametric displays the translated strings by searching for the same string from the list of string based messages that are loaded. For more information refer to the section on Localizing the Ribbon User Interface Created by the Creo Parametric TOOLKIT Applications on page 298.

You need a Creo Parametric development license to create, modify, or save the `toolkitribbonui.rbn` file. You cannot edit it manually. To save the ribbon user interface layout definition to the `toolkitribbonui.rbn` file:

1. Click **File ► Options**. The **Creo Parametric Options** dialog box opens.
2. Click **Customize Ribbon**.
3. In the **Customize the Ribbon** list, select a tab and create a new group in it or create a new tab and a group in it.
4. In the **Choose commands from** list, select **TOOLKIT Commands**. The commands created by the Creo Parametric TOOLKIT application are displayed.
5. Click **Add** to add the commands to the new tab or group.
6. Click **Import/Export ► Save the Auxiliary Application User Interface**. The modified layout is saved to the `toolkitribbonui.rbn` file located in the `text` folder within the Creo Parametric TOOLKIT application directory, that is, `<protk_app_dir>/text`
7. Click **OK**.

To Specify the Path for the Ribbon Definition File

You can rename the `toolkitribbonui.rbn` to another filename with the `.rbn` extension. To enable the Creo Parametric TOOLKIT application to read the ribbon definition file having a name other than `toolkitribbonui.rbn`, it must be available at the location `<protk_app_dir>/text/ribbon`. The function introduced in this section enables you to load the ribbon definition file from within a Creo Parametric TOOLKIT application.

Function Introduced:

- **ProRibbonDefinitionfileLoad()**

The function `ProRibbonDefinitionfileLoad()` loads a specified ribbon definition file from a default path into the Creo Parametric application. The input argument is as follows:

- *file_name* - Specify the name of the ribbon definition file including its extension. The default search path for this file is:
  - The working directory from where Creo Parametric is loaded.
  - `<application_text_dir>/ribbon`
  - `<application_text_dir>/language/ribbon`

> **Note**
>
> - The location of the application text directory is specified in the Creo Parametric TOOLKIT registry file.
> - A Creo Parametric TOOLKIT application can load a ribbon definition file only once. After the application has loaded the ribbon, calls made to the function `ProRibbonDefinitionfileLoad()` to load other ribbon definition files are ignored.

Loading Multiple Applications Using the Ribbon Definition File

Creo Parametric supports loading of multiple `.rbn` files in the same session. You can develop multiple Creo Parametric TOOLKIT applications that share the same tabs or groups and each application will have its own ribbon definition file. As each application is loaded, its `.rbn` file will be read and applied. When an application is unloaded, the containers and command created by its `.rbn` file will be removed.

For example, consider two Creo Parametric TOOLKIT applications, namely, `pt_geardesign` and `pt_examples` that add commands to the same group on a tab on the Ribbon user interface. The application `pt_geardesign` adds a

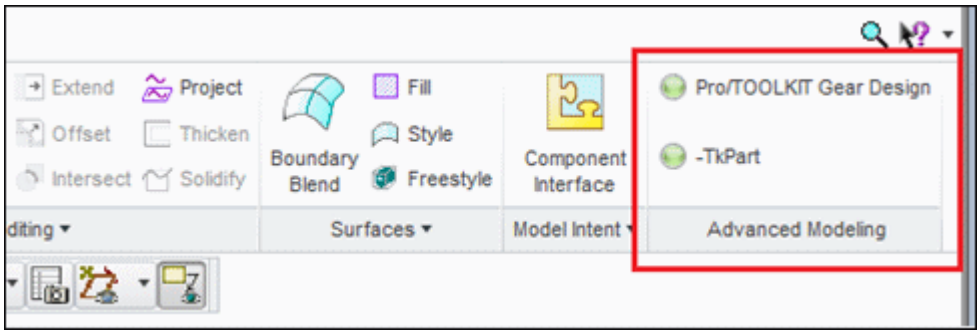
command **Pro/TOOLKIT Gear Design** to the **Advanced Modeling** group on the **Model** tab and the application `pt_examples` adds a command **TKPart** to the **Advanced Modeling** group on the **Model** tab. The ribbon definition file for each application will contain an instruction to create the **Advanced Modeling** group and if both the ribbon files are loaded, the group will be created only once and the two ribbon customizations will be merged into the same group.

That is, if both the applications are running in the same Creo Parametric session, then the commands, **Pro/TOOLKIT Gear Design** and **TKPart** will be available under the **Advanced Modeling** group on the **Model** tab.

Note

The order in which the commands will be displayed within the group will depend on the order of loading of the `.rbn` file for each application.

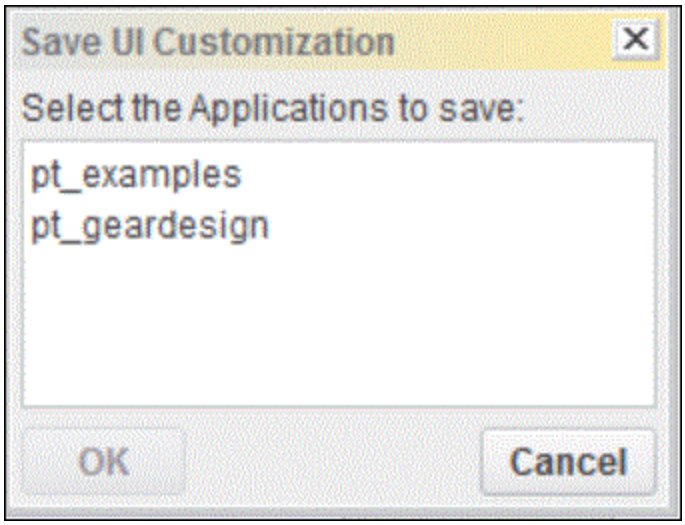
The following image displays commands added by two Creo Parametric TOOLKIT applications to the same group.



To save the customization when multiple applications are loaded:

1. Click **File ▸ Options**. The **Creo Parametric Options** dialog box opens.
2. Click **Customize Ribbon**.
3. In the **Customize the Ribbon** list, select a tab and create a new group in it or create a new tab and a group in it.
4. In the **Choose commands from** list, select **TOOLKIT Commands**. The commands created by the TOOLKIT application are displayed.
5. Click **Add** to add the commands to the new tab or group.
6. Click **Import/Export ▸ Save the Auxiliary Application User Interface**. The **Save UI Customization** dialog box opens.
7. Select a Creo Parametric TOOLKIT application and Click **Save**. The modified layout is saved to the `.rbn` file of the specified Creo Parametric TOOLKIT application.

The **Save UI Customization** dialog box is shown in the following image:



Localizing the Ribbon User Interface Created by Creo Parametric TOOLKIT Applications

The labels for the custom tabs, groups, and cascade menus belonging to the Creo Parametric TOOLKIT application can be translated in the languages supported by Creo Parametric. To display localized labels, specify the translated labels in the `ribbonui.txt` file and save this file at the location `<application_text_dir>/<language>`. For example, the text file for German locale must be saved at the location `<application_text_dir>/german/ribbonui.txt`.

Create a file containing translations for each of the languages in which the Creo Parametric TOOLKIT application is localized. The Localized translation files must use the UTF-8 encoding with BOM character for the translated text to be displayed correctly in the user interface. For more information on UTF-8 encoding, refer to Unicode Encoding on page 2077.

The format of the `ribbonui.txt` file is as shown below. Specify the following lines for each label entry in the file:

1. A hash sign (#) followed by the label, as specified in the ribbon definition file.
2. The label as specified in the ribbon definition file and as displayed in the ribbon user interface.
3. The translated label.
4. Add an empty line at the end of each label entry in the file.

For example, if the Creo Parametric TOOLKIT application creates a tab with the name TK_TAB having a group with the name TK_GROUP, then the translated file will contain the following:

```
#TK_TAB
TK_TAB
<translation for TK_TAB>
<Empty_line>
#TK_GROUP
TK_GROUP
<translation for TK_GROUP>
<Empty_line>
```

Tab Switching Events

If tab switching happens at run-time in Creo Parametric, use the notification functions to trigger a call back function to manage the changes due to the tab switching.

Functions Introduced:

- **ProNotificationSet()**
- **ProRibbonTabSwitchAction()**
- **ProNotificationUnset()**

Use the function `ProNotificationSet()` to assign a callback function to be called when the application switches from one tab to the other.

If the notification argument type is set to `PRO_RIBBON_TAB_SWITCH`, a registered call back function whose signature matches `ProRibbonTabSwitchAction()` is called. The “(Switch) from Tab” and “(Switch) to Tab” information is provided to the call back function, through the function arguments, whenever a Tab switch happens.

Use the function `ProNotificationUnset()` to cancel a notification.

For more information on using notifications see the chapter Event-driven Programming: Notifications.

Support for Legacy Pro/TOOLKIT Applications

The user interface for Creo Parametric 1.0 has been restructured to a ribbon user interface. This may affect the behavior of existing Pro/TOOLKIT applications that were designed to add commands to specific Pro/ENGINEER menus or toolbars. These menus or toolbars or both have been redesigned in Creo Parametric 1.0.

The commands added by the Pro/TOOLKIT applications appear on the Creo Parametric ribbon in the **Home** tab under the **TOOLKIT** group. When you open a model, the **TOOLKIT** group is on the **Tools** tab.

You can also arrange the commands added by the Pro/TOOLKIT applications under a new tab or an existing tab by customizing the ribbon using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. For a list of all the commands added by the Pro/TOOLKIT applications, follow this procedure:

1. Click **File ► Options**. The **Creo Parametric Options** dialog box opens.
2. Click **Customize Ribbon**.
3. In the **Choose commands from** list, select **TOOLKIT Commands**. All the commands added by legacy Pro/TOOLKIT applications are listed.

Note

Commands that have not been designated will not have an icon or will have a generic icon.

Refer to the Creo Parametric Help for more information on customizing the Ribbon.

Migration of Legacy Pro/TOOLKIT Applications

To migrate existing Pro/TOOLKIT applications to the Creo Parametric Ribbon user interface without compiling the source code:

1. Load the Pro/TOOLKIT applications in Creo Parametric so that the commands created in these applications are available in the **Customize Ribbon** user interface.
2. Modify the ribbon user interface layout and save the changes to the `toolkitribbonui.rbn`.
3. Copy the `toolkitribbonui.rbn` to the location `<application_text_dir>/ribbon`.
4. Reload Pro/TOOLKIT application or restart Creo Parametric. The `.rbn` file will be loaded along with the Pro/TOOLKIT application and the commands will be visible in the ribbon user interface if its accessibility will be visible.

17

User Interface: Menus, Commands, and Popupmenus

This chapter describes all the functions provided by Creo Parametric TOOLKIT to create and manipulate menus and buttons in the Creo Parametric ribbon user interface.

Refer to the chapter User Interface: Ribbon Tabs, Groups, and Menu Items on page 292 for more information. Also, refer to the Creo Parametric Help for more information on customizing the ribbon user interface.

Introduction

Using Creo Parametric TOOLKIT, you can supplement the Creo Parametric ribbon user interface.

Once the Creo Parametric TOOLKIT application is loaded, you can add a new group to an existing tab or create a new tab using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box in Creo Parametric. You will not be able to modify the groups that are defined by Creo Parametric. If the translated messages are available for the newly added tabs or groups, then Creo Parametric will use them by searching for the same string in the list of sting based messages loaded.

You can customize the ribbon user interface only for a particular mode in Creo Parametric. For example, if you customize the ribbon user interface and save it to the `toolkitribbonui.rbn` file in the Part mode, then on loading Creo Parametric the customized user interface will be visible only in the Part mode. To view a particular tab or group in all the modes, you must customize the ribbon user interface and save the`toolkitribbonui.rbn` file in each mode. Refer to the Creo Parametric Fundamentals Help for more information on customizing the ribbon.

When you are designing your Creo Parametric TOOLKIT application, you should carefully consider the context of the buttons that you add to the Creo Parametric User Interface (UI). Buttons specific to a particular mode (such as PART) should be located on the ribbon related to that mode. Buttons that initiate some action on a part, for example, should be located on the **PART** ribbon.

There are fundamental differences in the files and functions used to manipulate Ribbon and Menu-Manager menus. For this reason, this manual describes these subjects in separate sections.

Menu Buttons and Menus

The tabs in the Creo Parametric ribbon user interface contain groups composed of both buttons and submenus. Using Creo Parametric TOOLKIT, you can create similar structures in the Creo Parametric ribbon user interface. The object definitions are as follows:

- Push button—A named item in a group or menu that is used to launch a set of instructions. An example is the **Plane** button in the **Datum** group.

  Check button—An item in a group or menu that may be toggled on and off. An example is the **Plane Display** toggle in the **Model Display** group in the **View** tab.
- Radio group—An item in a group or menu that may be set to one and only one of any number of options. An example is the group of windows listed in the **Window** group at the **Windows** tab which allow you to switch between different windows.

- Command—A procedure in Creo Parametric that may be activated from a button.
- Command ID—An opaque pointer to a command, used as an input to other Creo Parametric TOOLKIT functions.
- Action command—A command which executes a set of instructions. Launched by push buttons.
- Option command—A command which executes a set of instructions based on the state of a UI component. Commands are launched by check buttons and radio groups.

Using the Trail File to Determine UI Names

Several functions dealing with UI components require the input of strings that Creo Parametric uses to identify commands and menu buttons.

To find the name of an action command, click the corresponding icon on the ribbon user interface and then check the last entry in the trail file. For example, for the save icon, the trail file will have the corresponding entry:

```
~ Command `ProCmdModelSave`
```

The command name for the save button is `ProCmdModelSave`. This string can be used as input to `ProCmdCmdIdFind()` to get the command ID.

Adding a PushButton

To add a button to the ribbon user interface, your Creo Parametric TOOLKIT application must do the following:

1. Define the action command to be initiated by the button. The action is defined in a function known as the “callback function.”
2. Designate the command using the function `ProCmdDesignate()`.
3. Add the button to the ribbon user interface using the using the **Customize Ribbon** tab in the **File ► Options** dialog box. This operation binds the added action to the button.

These procedures are described in the sections that follow.

Adding an Action to the Creo Parametric Ribbon

Function Introduced:

- **ProCmdActionAdd()**

The function `ProCmdActionAdd()` adds an action to Creo Parametric. This action can be later associated with a push button command in the Creo Parametric user interface. The syntax of this function is as follows:

```
ProError ProCmdActionAdd (
     char             *action_name,
     uiCmdCmdActFn     action_cb,
     uiCmdPriority     priority,
     uiCmdAccessFn     access_func,
     ProBoolean        allow_in_non_active_window,
     ProBoolean        allow_in_accessory_window,
     uiCmdCmdId       *action_id );
```

This function takes the following arguments:

- *action_name* —The name of the command as it will be used in Creo Parametric. This name must be unique, and it must occur only once in your applications or in Creo Parametric. To prevent conflicts, PTC recommends prepending or appending a unique identifier to your command names, similar to `ptc_openfile` or `openfile_ptc`.
- *action_cb*—The action function (callback function) that will be called when the command is activated by pressing the button, cast to a `uiCmdCmdActFn`:

  ```
  typedef int   (*uiCmdCmdActFn)   (
      uiCmdCmdId  command,
      uiCmdValue *p_value,
      void       *p_push_command_data
  );
  ```

  - *command*—Identifier of the action or option.
  - *p_value*—For options passed to `ValueGet` functions. Ignored for actions.
  - *p_push_command_data*Not implemented in this release.
- *priority*—The command priority. The priority of the action refers to the level of precedence the added action takes over other Creo Parametric actions.

  The available action priorities are defined in the enumerated type `uiCmdPriority`. The possible values are as follows:

  ```
  typedef int uiCmdPriority;
  #define uiCmdPrioDefault     ((uiCmdPriority) 0)
  #define uiProeImmediate      ((uiCmdPriority) 2)
  #define uiProeAsynch         ((uiCmdPriority) 3)
  #define uiProe2ndImmediate   ((uiCmdPriority) 5)
  #define uiProe3rdImmediate   ((uiCmdPriority) 6)
  #define uiCmdNoPriority      ((uiCmdPriority) 999)
  ```

  The following table describes the enumerated values in detail.

Value	Description
uiCmdPrioDefault	Normal priority actions Normal priority actions dismiss all other actions except asynchronous actions. Note that buttons of this priority can lead to the dismissal of Menu Manager menus. Dismissing these menus can result in unexpected behavior from functions that depend on the mode and the context of the Creo Parametric session. One example of a function which can exhibit unintended behavior is ProSelect() when selecting objects from an active simplified representation. Menu buttons should have lesser priority if they depend on the context of the Creo Parametric session.
uiProeImmediate, uiProe2ndImmediate, and uiProe3rdImmediate	Levels of immediate priority. Actions of each level of priority dismiss actions with lower level priorities.
uiProeAsynch	Asynchronous priority. Actions with asynchronous priority are independent of all other actions.

- *access_func*—The access function (callback function) that determines if the menu button should be available, unavailable, or hidden. Action accessibility refers to whether an added button is available for user selection. This function is called each time the button is displayed. The accessibility is evaluated based on the conditions pertaining at the time the button is pressed. The access function must be cast to a uiCmdAccessFn:

```
typedef uiCmdAccessState (*uiCmdAccessFn)
        (uiCmdAccessMode access_mode);
```

  The potential return values are listed in the enumerated type uiCmdAccessState:

  - ACCESS_REMOVE—The button is not visible, and the containing menus might also be removed from the menu, if all of the menu buttons in the containing menu possess an access function returning ACCESS_REMOVE.
  - ACCESS_INVISIBLE—The button is not visible.
  - ACCESS_UNAVAILABLE—The button is visible, but unavailable and cannot be selected.
  - ACCESS_DISALLOW—The button shows as available, but the command will not be executed when it is chosen.
  - ACCESS_AVAILABLE—The button is available and can be selected by the user.

Note

When you add a button, all the return values for the enumerated type `uiCmdAccessState` work as documented. However, when you add a button to the Creo Parametric ribbon user interface, and the access function returns the value `ACCESS_REMOVE` or `ACCESS_INVISIBLE`, these values are ignored. The values are treated as `ACCESS_UNAVAILABLE` instead. The button is visible, but is unavailable.

- *allow_in_non_active_window*—A `ProBoolean` determining whether or not to show this command in any non active window. A non-active window is a window that exists and contains a model, but that is no the active window in the Creo Parametric session. A window becomes active when the user chooses **View ► Activate**. This functionality is equivalent to changing the active window by selecting and activating a window using the pull-down menu of **Windows** command under the **View** tab in Creo Parametric.
- *allow_in_accessory_window*—A `ProBoolean` determining whether or not to show this command in an accessory window. An accessory window is smaller than a main Creo Parametric window and allows only the **File**>**Exit** command from the menu.
- *action_id*—The function will return a `uiCmdCmdId`, the command identifier. This identifier can be used in additional Creo Parametric function calls, such as, `ProCmdDesignate`.

Note

- The function `ProCmdActionAdd()` is executed only once per Creo Parametric session for each action. Subsequent calls to this function for a previously loaded action are ignored (therefore you cannot redefine an action within a Creo Parametric session).

Adding a Button to the Ribbon

You can add a button to the ribbon user interface using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. Refer to the chapter on User Interface: Ribbon Tabs, Groups, and Menu Items on page 292 for more information. Also, refer to the Creo Parametric Help for more information on customizing the Ribbon User Interface.

Adding a Check Button to the Ribbon User Interface

To add a check button to the ribbon user interface, your Creo Parametric TOOLKIT application must do the following:

1. Define the option command to be initiated by the button. The definition of this command includes the definition of three callback functions.
2. Designate the command using the function `ProCmdDesignate()`.
3. Add the check button to the ribbon user interface. This operation binds the added action to the button.

These procedures are described in the sections that follow.

Adding an Option Command to Creo Parametric—Check Button

Functions Introduced:

- **ProCmdOptionAdd()**
- **ProCmdChkbuttonValueGet()**
- **ProCmdChkbuttonValueSet()**

The function `ProCmdOptionAdd()` adds a command to Creo Parametric.

The syntax of this function is as follows:

```
ProError ProCmdOptionAdd (
      char              *option_name,
      uiCmdCmdActFn     option_cb,
      ProBoolean        boolean_operation,
      uiCmdCmdValFn     set_value_cb,
      uiCmdAccessFn     access_func,
      ProBoolean        allow_in_non_active_window,
      ProBoolean        allow_in_accessory_window,
      uiCmdCmdId        *option_id );
```

This function requires the following arguments:

- *option_name*—The name of the option command. This must be unique, in the same way as action command.
- *option_cb*—The action command to be executed when the check button is toggled, cast to a `uiCmdCmdActFn`. This function should include a call to `ProCmdChkbuttonValueGet()`, to determine the value of the check button.
- *boolean_operation*—Specifies whether or not the option has two values. Set this to `PRO_B_TRUE` for a check button.
- *set_value_cb*—The callback function that sets the value of the check button, cast to a `uiCmdCmdValFn`:

```
typedef int    (*uiCmdCmdValFn) (
    uiCmdCmdId  command,
    uiCmdValue *p_value
);
```

This function should include a call to `ProCmdChkbuttonValueSet()` to set the value of the check button when the UI is displayed or refreshed.

- *access_func*—The callback function that determines if the command is accessible.
- *allow_in_non_active_window*—A `ProBoolean` determining whether or not to show this command in any non-active window. A non-active window is a window that exists and contains a model, but that is not the active window in the Creo Parametric session. A window becomes active when the user activates a new window or opens a model in a new window.
- *allow_in_accessory_window*—A `ProBoolean` determining whether or not to show this command in an accessory window. An accessory window is smaller then a main Creo Parametric window and allows only the **File**>**Exit** command from the ribbon user interface.

The functions `ProCmdChkbuttonValueGet()` and `ProCmdChkbuttonValueSet()` provide access to the value of the check button. These functions require the option command value (provided by the callback functions as input), and the value is expressed as a `ProBoolean`.

Adding a Check Button to the Ribbon

You can add a check button to the ribbon user interface using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. Refer to the chapter on User Interface: Ribbon Tabs, Groups, and Menu Items on page 292 for more information. Also, refer to the Creo Parametric Help for more information on customizing the Ribbon User Interface.

Adding a Radio Button Group to the Ribbon

To add a radio button group to the ribbon user interface, your Creo Parametric TOOLKIT application must:

1. Define the option command to be initiated by the group of buttons. The definition of this command includes the definition of three callback functions.
2. Designate the command using the function `ProCmdDesignate()`
3. Add the radio button group to the ribbon user interface. This operation binds the added action to the button.

These procedures are described in the sections that follow.

Adding an Option Command to Creo Parametric—Radio Group

Functions Introduced:

- **ProCmdOptionAdd()**
- **ProCmdRadiogrpValueGet()**
- **ProCmdRadiogrpValueSet()**

The function `ProCmdOptionAdd()` is used to create the option command corresponding to the button group.

The arguments should be similar to the usage for creating the option command for a check button, with the following exceptions:

- *output_callback_function*—Must include a call to `ProCmdRadiogrpValueGet()` to determine the selected value in the radio group.
- *boolean_operations* —Must be `PRO_B_FALSE` for radio groups.
- *set_value_cb*—Must include a call to `ProCmdRadiogrpValueSet()` to set the value of the group upon redisplay of the radio group UI.

The functions `ProCmdRadiogrpValueGet()` and `ProCmdRadiogrpValueSet()` provide access to getting or setting the selected item in the group. They require the option command value (provided by the callback functions) as an input. The selected value is returned as a `ProMenuItemName` string.

Adding a Radio Button Group

You can add a button to the ribbon user interface using the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box. Refer to the chapter on User Interface: Ribbon Tabs, Groups, and Menu Items on page 292 for more information. Also, refer to the Creo Parametric Help for more information on customizing the Ribbon User Interface.

Manipulating Existing Commands

Functions Introduced:

- **ProCmdCmdIdFind()**
- **ProCmdAccessFuncAdd()**
- **ProCmdAccessFuncRemove()**
- **ProCmdBracketFuncAdd()**

The function `ProCmdCmdIdFind()` allows you to find the command ID for an existing command so you can add an access function or bracket function to the command. You must know the name of the command in order to find its ID. See section Using the Trail File to Determine UI Names on page 303 to determine UI names in order to determine the name of the command.

The functions `ProCmdAccessFuncAdd()` and `ProCmdAccessFuncRemove()` allow you to impose an access function on a particular command. (See function `ProCmdActionAdd()` for a description of access functions.) The Add function provides an `access_id`. This `ID` must be saved for later use when you deactivate the access function.

The function `ProCmdBracketFuncAdd()` allows the creation of a function that will be called immediately before and after execution of a given command. this function would be used to add company logic to the start or end (or both) of an existing Creo Parametric command. It could also be used to cancel an upcoming command. This function is declared as:

```
ProError ProCmdBracketFuncAdd (
     uiCmdCmdId         cmd_id,
     uiCmdCmdBktFn      bracket_func,
     char              *bracket_func_name,
     void             **pp_bracket_data );
```

The function takes the following arguments:

- *cmd_id*—The command identifier.
- *bracket_func*—The callback function to be called before and after the command, cast to a `uiCmdCmdBktFn`:

  ```
  typedef int    (*uiCmdCmdBktFn)
     ( uiCmdCmdId   command,
       uiCmdValue  *p_new_value,
       int          entering_command,
       void       **pp_bracket_data);
  ```

  The entering command argument will be `1` before execution and `0` after. If the operation is before the upcoming command execution, and you want to cancel the upcoming command execution, return `0`. Otherwise, return non-zero.
- *bracket_func_name*—The name of the bracket function.
- *pp_bracket_data*—A `void**` containing data to be passed to the bracket function.

Designating Commands

Using Creo Parametric TOOLKIT you can designate Creo Parametric commands. These commands can later appear in the Creo Parametric ribbon user interface.

In Creo Parametric TOOLKIT you can set a button to refer to a command and subsequently drag this button on to the Creo Parametric ribbon user interface. When the button is clicked, the command is executed.

To add a command, your Creo Parametric TOOLKIT application must do the following:

1. Define or add the command to be initiated on clicking the icon.
2. Optionally designate an icon button to be used with the command defined by you.
3. Designate the command (icon) to appear in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box.

> **Note**
>
> Refer to the chapter on User Interface: Ribbon Tabs, Groups, and Menu Items on page 292 for more information. Also, refer to the Creo Parametric Help for more information on customizing the Ribbon User Interface.

4. Save the configuration in Creo Parametric so that changes to the ribbon user interface appear when a new session of Creo Parametric is started.

Adding the Command

Functions Introduced:

- **ProCmdActionAdd()**
- **ProCmdOptionAdd()**

The functions `ProCmdActionAdd()` and `ProCmdOptionAdd()` allow you to define or register a Creo Parametric command. See the section Adding an Action to the Creo Parametric Ribbon on page 303 for more information on the function `ProCmdActionAdd()` and the section Adding an Option Command to Creo Parametric—Check Button on page 307 for more information on the function `ProCmdOptionAdd()`.

Providing the Icon

Function Introduced:

- **ProCmdIconSet()**

The function `ProCmdIconSet()` allows you to provide an icon to be used with the command you created. The function adds the icon to the Creo Parametric command. The function takes the command identifier as one of the inputs and the name of the icon file, including the extension as the other input. A valid format for

the icon file is a standard `.GIF`, `.JPG`, or `.PNG`. PTC recommends using `.PNG` format. All icons in the `Creo Parametric` ribbon are either 16x16 (small) or 32x32 (large) size. The naming convention for the icons is as follows:

- Small icon—`<iconname.ext>`
- Large icon—`<iconname_large.ext>`

Note

The legacy naming convention for icons `<icon_name_icon_size.ext>` will not be supported in future releases of Creo Parametric. The icon size was added as a suffix to the name of the icon. For example, the legacy naming convention for small icons was `iconname16X16.ext`. It is recommended to use the standard naming conventions for icons, that is, `iconname.ext` or `iconname_large.ext`.

The application searches for the icon files in the following locations:

- `<creo_loadpoint>\<datecode>\Common Files\text\resources`
- `<application_text_dir>\resource`
- `<application_text_dir>\<language_dir>/resource`

  The location of the application text directory is specified in the Creo Parametric TOOLKIT registry file.

The Creo Parametric button is replaced with the icon image.

Commands that do not have an icon assigned to them display the button label.

You may also use this function to assign a small icon to a button.

Designating the Command

Function Introduced:

- **ProCmdDesignate()**
- **ProCmdRadiogrpDesignate()**
- **ProCmdAlwaysAllowValueUpdate()**

The function `ProCmdDesignate()` allows you to designate a command or check button to be available in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog of Creo Parametric.

After a Creo Parametric TOOLKIT application has used the function `ProCmdDesignate()` on a command, the user can add the button associated with this command into the Creo Parametric ribbon user interface.

If this function is not called, the button will not be visible in the **TOOLKIT Commands** list in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog of Creo Parametric.

The syntax of the function `ProCmdDesignate()` is:

```
ProError ProCmdDesignate ( uiCmdCmdId cmd_id,
                           ProMenuItemLabel button_label,
                           ProMenuLineHelp one_line_help,
                           ProMenuDescription description,
                           ProFileName msg_file);
```

The arguments to this function are:

- *cmd_id*—The command identifier.
- *button_label*—The message string that refers to the icon label. This label (stored in the message file) identifies the text seen when the button is displayed. If the command is not assigned an icon, the button_label string appears on the button by default.
- *one_line_help*—The one-line Help for the icon. This label (stored in the message file) identifies the help line seen when the mouse moves over the icon.
- *description*—The message appears in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog and also when "Description" is clicked in Creo Parametric.
- *msg_file*—The message file name. All the labels including the one-line Help labels must be present in the message file.

> **Note**
>
> This file must be in the directory `<text_path>/text` or `<text_path>/text/<language>`.

The function `ProCmdRadiogrpDesignate` designates the radio button to be available in the **TOOLKIT Commands** list in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog box of Creo Parametric. The input arguments of this function are:

- *option_id*—The option identifier.
- *number_radio_group_items*—Specifies the number of options in the radio group
- *radio_group_items*—Specifies an array of items in the radio group.

- *radio_group_labels*—Specifies the labels for the radio buttons. This label (stored in the message file) identifies the text seen when the button is displayed. If the command is not assigned an icon, the label string appears on the menu by default.
- *one_line_helps*—The one-line Help for the icon. This label (stored in the message file) identifies the help line seen when the mouse moves over the icon.
- *radio_group_icons*—Specifies an array of icon file names, including the extension. A valid format for the icon file is a standard `.GIF`, `.JPG`, or `.PNG`. PTC recommends using `.PNG` format. All icons in the `Creo Parametric` ribbon are either 16x16 (small) or 32x32 (large) size. The naming convention for the icons is as follows:
  - Small icon— `<icon_name_16X16.ext>`
  - Large icon— `<icon_name_32X32.ext>`

  The application searches for the icon files in the following locations:
  - `<creo_loadpoint>\<datecode>\Common Files\text\resources`
  - `<application_text_dir>\resource`
  - `<application_text_dir>\<language_dir>\resource`

    The location of the application text directory is specified in the Creo Parametric TOOLKIT registry file.
- *description*—The message appears in the **Customize Ribbon** tab in the **Creo Parametric Options** dialog and also when "Description" is clicked in Creo Parametric.
- *msg_file*—The message file name. All the labels including the one-line Help labels must be present in the message file.

> **Note**
>
> This file must be in the directory `<text_path>/text` or `<text_path>/text/<language>`.

The function `ProCmdAlwaysAllowValueUpdate()` allows the value of the command to be updated always, even when the command is not accessible. By default, `set_value_cb` is called only when the command is accessible. The input arguments follow:

- *cmd_id*—The command identifier.
- *allow*—Specify as `PRO_B_TRUE` for the value of the command to be updated always. Specify it as `PRO_B_FALSE`, only when the command is accessible.

Placing the Button

Once the button has been created using the functions discussed, place the button on the Creo Parametric ribbon user interface. Refer to the chapter on User Interface: Ribbon Tabs, Groups, and Menu Items on page 292 for more information. Also, refer to the Creo Parametric Help for more information on customizing the Ribbon User Interface.

Example 1: Designating a Command

The example code in the file `UgMain.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main` illustrates how to designate a command to be available for placement as a button.

Popup Menus

Creo Parametric provides shortcut menus that contain frequently used commands appropriate to the currently selected items. You can access a shortcut menu by clicking the right mouse button (RMB) after selecting an item. Shortcut menus are accessible in:

- Graphics window
- Model tree
- Some dialog boxes
- Any area where you can perform an object-action operation by selecting an item and then choosing a command to perform on the selected item.

Creo Parametric TOOLKIT provides different procedures to add custom buttons to popup menus, depending on the buttons' context. To add to the model tree popup menu, use the procedure described in Adding a Button to the Model Tree Popup Menu on page 320. To add a popup menu to a custom dialog box, see the User Interface: Dialogs on page 344 chapter (you cannot modify the popup menus in an existing UI dialog box). To add to a graphics window popup menu, refer to Adding a Popup Menu to the Graphics Window on page 315.

Adding a Popup Menu to the Graphics Window

Different popup menus can be activated during a given session of Creo Parametric. Every time the Creo Parametric context changes (by opening a different model type, by entering different tools, or by entering special modes like "Edit") a different popup menu is created. When Creo Parametric moves to the next context, the popup menu may be destroyed.

Because of this, Creo Parametric TOOLKIT applications must attach a button to the popup menu during initialization of the popup menu. The Creo Parametric TOOLKIT application is notified each time a particular popup menu is created, which then allows the user to add to the popup menu.

Use the following procedure to add items to graphics window popup menus:

1. Obtain the name of the existing popup menus to which you want to add a new menu using the trail file.
2. Register the Creo Parametric TOOLKIT notifications for creation and destruction of popup menus.
3. Create commands for the new popup menu items.
4. Implement access functions to provide visibility information for the items.
5. When the notification is called for the desired popup menu, add the custom buttons to the menu.
6. When the destroy notification is called, free the associated memory for the custom buttons.

The following sections describe each of these steps in detail. You can add push buttons, check buttons, and cascade menus to the popup menus. You can add popup menu items only in the active window. You cannot use this procedure to remove items from existing menus. To remove items using an access function see the section on Manipulating Existing Commands on page 309.

Using the Trail File to Determine Existing Popup Menu Names

The trail file in Creo Parametric contains a comment that identifies the name of the popup menu if the configuration option `auxapp_popup_menu_info` is set to `yes`.

For example, the popup menu, **Edit Properties**, has the following comment in the trail file:

```
~ Close `rmb_popup` `PopupMenu`
~ Command `ProCmdEditProperties`
```

Registering Notifications to Create and Destroy Popup Menus

Popup menus are created at runtime in Creo Parametric and only when the required menus exist in the active window. Hence it is not possible to pre-register the custom buttons. Notification functions notify applications to add a new popup menu. For more information on using notifications see the chapter Event-driven Programming: Notifications on page 2010.

Functions Introduced:

- **ProNotificationSet()**
- **ProPopupmenuCreatePostAction()**
- **ProPopupmenuDestroyPreAction()**

If the notification value argument type is set to `PRO_POPMENU_CREATE_POST`, a registered callback function whose signature matches `ProPopupmenuCreatePostAction()` is called. This function is called after the popup menu is created internally in Creo Parametric and must be used to assign application-specific buttons to the popup menu.

If the notification value argument type is set as `PRO_POPUPMENU_DESTROY_PRE`, a registered callback notification function `ProPopupmenuDestroyPreAction()` is called before the popup menu is destroyed. Use this function to free memory allocated by the application for the custom buttons in the popup menu.

Accessing the Popup Menus

The functions described in this section provide the name and ID of the popup menus that are used to access these menus while using other functions.

Functions Introduced:

- **ProPopupmenuIdGet()**
- **ProPopupmenuNameGet()**

The function `ProPopupmenuIdGet()` returns the popup menu ID for a given popup menu name. The popup menu ID is required for the functions that add buttons to the popup menu. IDs are dependent on the context of Creo Parametric and are not maintained between sessions.

The function `ProPopupmenuNameGet()` returns the name of the popup menu assigned to a given ID.

Creating Commands for the New Popup Menu Buttons

Functions Introduced:

- **ProCmdActionAdd()**
- **ProCmdOptionAdd()**
- **ProCmdCmdIdFind()**

The functions `ProCmdActionAdd()` or `ProCmdOptionAdd()` are used to create commands for the popup menus. Only push buttons (action commands) and check buttons (option commands) are supported in popup menus. Commands with

a given name are created only once in a session of Creo Parametric. Hence PTC recommends that you create the required commands in the `user_initialize()` function of the application.

Use `ProCmdCmdIdFind()` to obtain the command ID for the command (in the notification callback for `PRO_POPUPMENU_CREATE_POST`) to add the button to the popup menu.

Checking the Access State of a Popup Menu Item

Functions Introduced:

- **ProPopupmenuAccessFunction()**

A popup menu uses an additional access function to determine whether the popupmenu must be visible based on the currently selected items. Use the function whose signature matches `ProPopupmenuAccessFunction()` to set the access state of the button in the popup menu.

The syntax for this function is as follows:

```
typedef uiCmdAccessState (*ProPopupmenuAccessFunction)
(uiCmdCmdId command,
 ProAppData appdata,
 ProSelection* sel_buffer);
```

The last argument contains an array of selected items that are used to determine the visibility of the popup menu button. It is PTC standard practice to remove popup menu buttons using `ACCESS_REMOVE` instead of graying them out using `ACCESS_UNAVAILABLE` when unusable item types have been selected. This is to minimize the size of the popup menu.

Adding Creo Parametric Popup Menus

Functions Introduced:

- **ProPopupmenuButtonAdd()**
- **ProPopupmenuCascadebuttonAdd()**

Use `ProPopupmenuButtonAdd()` to add a new item to a popup menu. The input arguments are:

- *menu_ID*—Specifies the ID of the popup menu obtained from `ProPopupmenuIdGet()`.
- *position*—Specifies the position in the popup menu at which to add the menu button. Pass `PRO_VALUE_UNUSED` to add to the bottom of the menu.
- *button_name*—Specifies the name of the added button. The button name is placed in the trail file when the user selects the menu button. For more

information on valid characters that you can use to specify the name, refer to the section Naming Convention for UI Components on page 349.

- *button_label*—Specifies the message that refers to the button label. This label identifies the text seen when the button is displayed. To localize this text, obtain and pass a string from `ProMessageToBuffer()`.

> **Note**
>
> The labels and the text added using the `ProCmdDesignate()` function duplicate existing messages that are previously added in the Creo database. To display the correct label and text message, you can use a prefix or a suffix with the message names that will identify your Creo Parametric TOOLKIT application. You should avoid using generic names of Creo Parametric TOOLKIT buttons such as Point, Arc, Circle, Ellipse in the labels and text.

- *button_helptext*—Specifies the help message associated with the button. This label acts as a keyword that identifies the help text in the message file. To localize this text, obtain and pass a string from `ProMessageToBuffer()`.
- *cmd_ID*—Specifies the command identifier for the action or option.
- *access_status*—Specifies the callback function used to determine the visibility status of the added button.
- *appdata*—Specifies the user application data.

Use the function `ProPopupmenuCascadebuttonAdd()` to add a new cascade menu to an existing popup menu.

The input arguments are:

- *menu_ID*—Specifies the ID of the popup menu obtained from `ProPopupmenuIdGet()`.
- *position*—Specifies the position in the menu at which to add the cascade button. Pass `PRO_VALUE_UNUSED` to add to the bottom of the menu.
- *cascade_menu_name*—Specifies the name of the cascade menu. The name is placed in the trail file when the user selects the menu button. For more information on valid characters that you can use to specify the name, refer to the section Naming Convention for UI Components on page .
- *cascade_menu_label*—Specifies the message that refers to the cascade menu label. This label identifies the text seen when the menu is displayed. To localize this text, obtain and pass a string from `ProMessageToBuffer()`.

- *cascade_menu_helptext*—Specifies the help message associated with the cascade menu. This label acts as a keyword that identifies the help text in the message file. To localize this text, obtain and pass a string from `ProMessageToBuffer()`.
- *access_status*—Specifies the callback function used to determine the visibility status of the added item.
- *appdata*—Specifies the user application data.

The output argument is *casc_menuId*, the menu ID of the cascade menu.

Adding a Button to the Model Tree Popup Menu

To add a button to the ribbon user interface, refer to the section Menu Bar Buttons and Menus on page 302. Ensure that the following conditions are met:

- The menu name used should be "ActionMenu."
- The command access function should be configured to read the selection buffer using `ProSelbufferSelectionsGet()`. If the buffer is inactive or empty, the access function should make the menu item invisible.

Example 2: Assigning the Creo Parametric command to popup menus

The example in the file `UgPopupmenus.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main` demonstrates how to assign a Creo Parametric command to the popup menus in the graphics window and the model tree. It adds the example `UserAsmcompConstraintsHighlight()` to the popup menu (see the chapter on Assembly: Basic Assembly Access on page 1130 for details). The popup menus use an access function to check the currently selected item in the selection buffer that is an assembly component. If it is not an assembly component, the button will be removed from the menu. The example adds the popup menu button using `ProPopupmenuButtonAdd()` if the menu name is the name of the current main popup menu in assembly mode.

Menu Manager Buttons and Menus

Very few modes in Creo Parametric display a menu manager. For example, the **ASM PROCESS** menu is displayed only when Creo Parametric is in the Process Plan Assembly mode, which occurs when a Process-Plan Assembly has been created or retrieved. Modifying and supplementing the menu manager interface is fundamentally different from similar operations on the ribbon.

This section describes the files and functions necessary to manipulate the menu manager buttons and menus of Creo Parametric. This section covers the following topics:

Menu Files

Menu files enable you to specify your own text for the name of a menu button and the one-line help text that appears when you place the cursor over that button, along with translations for both of these.

Creo Parametric looks for the Creo Parametric TOOLKIT menu files in the following locations:

- The current Creo Parametric startup directory
- The subdirectory `text/menus` under the directory named by the `text_dir` statement in the registry file

PTC recommends that during development you place your menu files in `text/menus` under your working directory and specify the following registry file entry:

```
text_dir .
```

Names and Contents of Menu Files

There are two conventional extensions used in naming menu files:

- `.mnu`—Files that describe complete menus
- `.aux`—Files that describe new buttons to be added to existing Creo Parametric menus

The following restrictions apply to file names:

- The name must be unique through out Creo Parametric.
- The name must have no more than 30 characters, including the extension.

To find out what menu file names are used by Creo Parametric, look in the Creo Parametric menu directory at `<creo_loadpoint>\<datecode>\Common Files\text\<language_dir>\menus`.

When you create an `.aux` file to extend an existing Creo Parametric menu, use the same file name root as Creo Parametric used for that menu.

Syntax and Semantics of Menu Files

The two types of files—`.mnu` and `.aux`—have identical formats.

The format consists of groups of three lines (one group for each menu button) and a group at the top for the menu title. The title group contains the menu title on the first line, and then two blank lines.

The menu title is the name that appears at the top of the menu when you run Creo Parametric in English. The menu title is also used to refer to the menu from your Creo Parametric TOOLKIT code, so it is essential that this name is unique in all Creo Parametric menus. For example, if you are writing an `.aux` file to add buttons to a Creo Parametric menu, make sure you use the title that appears in the corresponding `.mnu` file in Creo Parametric. If you are creating a new menu, make sure that the title you use has not already been used in Creo Parametric.

If the menu title is followed by a second word, Creo Parametric displays the second word instead of the first one. This is how a translation is provided. If there is no second word, Creo Parametric displays the first word.

Each menu button group consists of the following three lines:

- Button name—If the button name as it appears on the Creo Parametric screen contains spaces, each space must be replaced by the character # in the menu file. If the button name is followed by another name, separated by white space, the second name will be what is actually displayed.

  The first name is still used to refer to the button from your Creo Parametric TOOLKIT code. The second provides an optional translation of that button name.
- One-line Help—This is a single line of text that explains what the menu button does. When you place the mouse pointer on the menu button, Creo Parametric displays the one-line Help text in the Message Window.
- Alternate Help—If this line is not blank (or does not start with the comment character “#”), it will be used in place of the one-line Help. This provides a translation of the Help message.

Example 3: Sample Menu File

The following example code shows the menu file you would create to add a new button, **Check Part**, to the Creo Parametric **PART** menu.

```
 Menu file "part.aux":
[Start of file on next line]
PART
<blank line>
<blank line>
Check#Part
Check the validity of the current part.
<blank line>
[End of file on previous line]
```

Example 4: Adding Alternate Names and Help Text to a Button

This example code creates an alternate button name and Help text for the previous example.

```
Menu file "part.aux":
[Start of file on next line]
PART
<blank line>
<blank line>
Check#Part DRC#Check
Check the validity of the current part.
Perform a DRC (Design Rule Check) on the part.
[End of file on previous line]
```

Adding a Menu Button

Functions Introduced:

- **ProMenuFileRegister()**
- **ProMenuAuxfileRegister()**
- **ProMenubuttonActionSet()**
- **ProMenubuttonGenactionSet()**

When you add a new button to an existing menu in `user_initialize()`, you are modifying the Creo Parametric definition of the menu in its memory before that menu has been used by Creo Parametric, and therefore before Creo Parametric has loaded it from its menu file. You must call the function `ProMenuFileRegister()` to tell Creo Parametric to load its own menu file before you can add your own buttons.

To add a button to a menu, first write a menu file, and then add the following calls to `user_initialize()`:

1. Load the Creo Parametric menu into memory, using `ProMenuFileRegister()`.
2. Add the buttons in your menu file to the menu, using `ProMenuAuxfileRegister()`.
3. Define the actions of the new buttons, using `ProMenubuttonActionSet()`.

Calling ProMenuFileRegister()

The input arguments to `ProMenuFileRegister()` are as follows:

- `ProMenuName` *menuname*—The unique title of the menu that appears as the first word on the first line of the menu file and on the heading of the menu on the screen when you run Creo Parametric in English. This argument is case-insensitive.
- `ProMenufileName` *filename*—The name of the menu file, including the extension but not the directory.

The function outputs the integer identifier of the menu, which you do not normally need. If the function fails for some reason (for example, the menu file did not exist), it returns `PRO_TK_GENERAL_ERROR`. If you call this function a second time on the same menu file, it has no effect.

Calling ProMenuAuxfileRegister()

This function has the same arguments and return value as `ProMenuFileRegister()`. Instead of loading a new menu into memory, the function adds the buttons in the file to a menu already in memory.

Calling ProMenubuttonActionSet()

The first three arguments to `ProMenubuttonActionSet()` are as follows:

- `ProMenuName` *menuname*—The title of the menu that contains the button.
- `ProMenubuttonName` *button*—The first name for the button in the menu file (not the second, which provides the translation), but with spaces instead of pound signs (#). This argument is case-insensitive.
- `ProMenubuttonAction` *action*—A pointer to the Creo Parametric TOOLKIT callback function to be called when the user selects this menu button. To pass a pointer to a function, supply the name of the function without the following parentheses. If your function does not precede the call to `ProMenubuttonActionSet()` in the same file, you must add a declaration of it to show the compiler that this is a function.

The other two arguments, *app_data* and *app_int*, are optional arguments to your command function. These arguments enable your command function to be more flexible in what it does. If you do not want to use *app_data* and *app_int*, supply the values `NULL` and `0`, respectively.

Sample declarations and the use of the optional arguments are shown in Example 5: Adding a Button to the Creo Parametric Ribbon on page 325; Example 6: Defining a New Menu that Closes Itself on page 326; and Example 7: Defining a New Menu the User Must Close on page 327.

Example 5: Adding a Button to the Creo Parametric Ribbon

The example code in the file `UgMenuMenuButtonAdd.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_menu` adds the button **Check Part** to the Creo Parametric **PART** tab. The example uses the menu file from the previous examples.

New Menus

Functions Introduced:

- **ProMenuProcess()**
- **ProMenuDelete()**
- **ProMenuCreate()**
- **ProMenuHold()**
- **ProMenuDeleteWithStatus()**

Creo Parametric TOOLKIT enables you to create new menus. Defining a new menu differs from adding buttons to an existing menu in the following ways:

- The menu file you supply should end in `.mnu`, not `.aux`. (It has the same syntax, though.)
- You do not need to call `ProMenuAuxfileRegister()` because the whole menu is defined in a single menu file.
- You need to define an exit action for the menu, in addition to an action for each button on the menu.
- You can either specify the new menu in `user_initialize()` or you can set up the new menu locally before you use it.

Exit Actions

You must not only tell the menu manager inside Creo Parametric which function to call for each button on your menu, but also which function to call if the user selects a button on another menu. This function is called an exit action because it is often used to close the menu.

Note

If you do not define an exit action, Creo Parametric's behavior is undefined if the user selects from another menu.

There are two types of exit action:

- Nothing—The menu selection is ignored. This is useful if you want the user to take some definite action before leaving the current menu.
- Close the current menu—The menus unwind to the level of the menu selected and the selected command is entered. This is the usual way to leave a menu.

Defining a New Menu

To define a new menu, first write a menu file. Before you need to use the menu, add the following calls to your Creo Parametric TOOLKIT program:

1. Load the Creo Parametric menu into memory, using `ProMenuFileRegister()`.
2. Define the actions of the new buttons, using the function `ProMenubuttonActionSet()`.
3. Define the exit action of the new menu, using the functions `ProMenubuttonActionSet()` and one of the exit action functions described in the following section.

Example 6: Defining a New Menu that Closes Itself

This example code defines a new menu, MYMENU, that closes itself using the function `ProMenuDelete()`.

```
[Start of file on next line]
MYMENU
<blank line>
<blank line>
Partial#Check
Perform a partial check on the part.
<blank line>
Full#Check
Perform a full check on the part.
<blank line>
[End of file on previous line]
```

The following code sets up the menu:

```
    int menuId;
    ProMenuFileRegister ("mymenu", "mymenu.mnu", &menuId);
    ProMenubuttonActionSet ("mymenu", "Partial Check", ProCheckPart,
        NULL, 0);
```

```
ProMenubuttonActionSet ("mymenu", "Full Check", ProCheckPart, NULL,
    1);
ProMenubuttonActionSet ("mymenu", "Quit Checks",
    (ProMenubuttonAction)ProMenuDelete, NULL, 0);
ProMenubuttonActionSet("mymenu", "mymenu",
    (ProMenubuttonAction)ProMenuDelete, NULL, 0);
```

Example 7: Defining a New Menu the User Must Close

In the following example code, the user has to close `MYMENU`.

```
int menuId;
ProMenuFileRegister ("mymenu", "mymenu.mnu", &menuId);
ProMenubuttonActionSet ("mymenu", "Partial Check", ProCheckPart,
    NULL, 0);
ProMenubuttonActionSet ("mymenu", "Full Check", ProCheckPart,
    NULL, 1);
ProMenubuttonActionSet ("mymenu", "Quit Checks",
    (ProMenubuttonAction)ProMenuDelete, NULL, 0);
ProMenubuttonActionSet ("mymenu", "mymenu",
    (ProMenubuttonAction)ProMenuHold, NULL, 0);
```

Defining an Exit Action

To define an exit action, make an extra call to `ProMenubuttonActionSet()`, but instead of the button name (the third argument), specify the menu name.

If you want the menus to unwind and the new command to be entered, use `ProMenuDelete()` as the action function.

If you want the selection to be ignored, use the function `ProMenuHold()` as the exit action. If you use this function, you must provide some other exit route for the menu. For example, you can specify an explicit menu button (such as **Done**) whose command function calls `ProMenuDelete()`.

If you want to perform some additional action in these cases (such as sending a warning to the user), you can provide your own exit function that performs the action and then calls `ProMenuHold()`.

Using a New Menu

After you have defined your new menu, you need to know how to use it. This is normally done inside the command function of another menu button.

To Use a New Menu

1. Display the menu, using `ProMenuCreate()`.
2. Make the menu active so the user can select from it, using `ProMenuProcess()`.

Calling ProMenuCreate()

The first argument to `ProMenuCreate()` is either `PROMENUTYPE_MAIN` or `PROMENUTYPE_SUB`. The usual choice is `PROMENUTYPE_MAIN` (see the section Submenus on page 331 for detailed information about submenus). The second argument is the title of the menu. The last argument is the identifier of the displayed menu.

Calling ProMenuProcess()

The function `ProMenuProcess()` takes a single input argument—the title of the menu. If the menu is the last one displayed, you can pass an empty string. The return value is meaningful only if you use the function `ProMenuDeleteWithStatus()` as the exit action for the menu.

The function `ProMenuProcess()` returns only when the menu is closed, as the result of a call to either `ProMenuDelete()` or `ProMenuDeleteWithStatus()`. The following is true for any code following the call to `ProMenuProcess()`:

1. The code does not get executed until the menu is closed.
2. The code gets executed before any command that causes an exit from the menu. When the user closes a menu by selecting another command, that command is put into the input buffer and is not executed until control passes from your application back to Creo Parametric.

Example 8: Using a New Menu

The following example code shows how to use the functions `ProMenuProcess()` and `ProMenuCreate()`. The example builds on the previous examples.

```
int action, menuId;
.
.
.
ProMenuCreate (PROMENUTYPE_MAIN, "mymenu", &menuId);
ProMenuProcess ("", &action);
.
.
```

Creating a Menu for Selecting a Single Value

Function Introduced:

- **ProMenuDeleteWithStatus()**
- **ProMenubuttonActionSet()**
- **ProMenuProcess()**

Use of ProMenubuttonActionSet() Final Arguments

The two last arguments of `ProMenubuttonActionSet()` are *app_data*, of type `ProAppData` and *app_int*, of type integer. These arguments are passed directly to your callback function when it is invoked. Because Creo Parametric TOOLKIT and Creo Parametric do not look at these arguments, you can use them for any information that you want to pass to or from your function.

Example 9: Creating a Menu that Selects a Value on page 329 uses the final argument of `ProMenubuttonActionSet()` to distinguish between several menu buttons that share the same command function. Inside the command function, this value appears as the second argument. It is used to determine which button was selected and then perform the appropriate action. The command function does not use the fourth argument of `ProMenubuttonActionSet()`, but includes a dummy first argument of type `ProAppData` to match it, so that the second argument is received correctly.

Returning a Value from ProMenuProcess()

The function `ProMenuDelete()` closes the current menu and causes control to return from the call to `ProMenuProcess()` that made that menu active. If you want to close the menu under more than one condition and react to that condition in the code that follows the return from `ProMenuProcess()`, use `ProMenuDeleteWithStatus()` instead of `ProMenuDelete()`. The `ProMenuDeleteWithStatus()` function takes a single integer argument, which is the value returned by `ProMenuProcess()`.

Example 9: Creating a Menu that Selects a Value

The example code in the file `UgMenuValueSelect.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_menu` shows several new techniques for using the menu functions. This example shows how to use `ProMenuDeleteWithStatus()` and uses more of the arguments to `ProMenubuttonActionSet()`.

Compound Menus

Function Introduced:

- **ProCompoundmenuCreate()**

The `ProCompoundmenuCreate()` function enables you to take an array of previously loaded menu names and append them together into one menu.

To Create a Compound Menu:

1. Specify which submenus to include in the compound menu, as follows:
   ```
   static char **compound_menu = {"MENU_1","MENU_2",  "MENU_3", ""};
   ```
2. Load the actions on the buttons.

3. Set the button visibility and accessibility.
4. Generate the compound menu, as follows:
   ```
   ProCompoundmenuCreate (compound_menu, n_submenus);
   ```
5. Get user input, as follows:
   ```
   ProMenuProcess (compound_menu[0], action);
   ```

Preempting Creo Parametric Commands

Functions Introduced:

- **ProMenubuttonPreactionSet()**
- **ProMenubuttonPostactionSet()**

In addition to adding your own menus and menu buttons, it is sometimes useful to be able to modify the effect of an existing Creo Parametric menu button. The function `ProMenubuttonPreactionSet()` enables you to call your function before the Creo Parametric command is executed. If the operation is before the upcoming command execution, and you want to cancel the upcoming command execution, return 1. Otherwise, return zero.

You could also cancel the Creo Parametric command, so only your function gets called. Similarly, the function `ProMenubuttonPostactionSet()` enables you to call your function after the Creo Parametric command is executed.

You can use the `ProMenubuttonPreactionSet()` function to protect certain commands, so that the user can use them only under certain circumstances specified by your Creo Parametric TOOLKIT application. For example, you may want to prevent the user from saving a model unless it has passed a certain validity check.

Calling ProMenubutton*actionSet()

The functions `ProMenubuttonPreactionSet()` and `ProMenubuttonPostactionSet()` have the same arguments as `ProMenubuttonActionSet()`. The function `ProMenubuttonPreactionSet()` inserts your function before an existing Creo Parametric command instead of assigning it to a new button. The function `ProMenubuttonPostactionSet()` inserts your function after an existing Creo Parametric command.

Because you are changing the definition of the menu in Creo Parametric, you must make sure the menu is loaded into memory first, by calling `ProMenuFileRegister()`.

If the command function you load returns the value 0, the Creo Parametric command for that menu button will be executed immediately. If your function returns any other value, the Creo Parametric command will not be performed.

Example 10: Asking for Confirmation on Quit Window

The example code in the file `UgMenuConfirmGet.c` located at `<creo_toolkit_loadpoint>protk_appls/pt_userguide/ptu_menu` shows how to use `ProMenubuttonPreactionSet()` to ask the user to confirm a selection. The example uses `ProMenubuttonPreactionSet()` to protect **Quit Window**

Submenus

Function Introduced:

- **ProMenuCreate()**

All the menus described so far have been main menus. The other type of menu is called a submenu. A submenu differs from a main menu in the following ways:

- A submenu is active at the same time as the menu above it. Selecting from the menu above does not close the submenu.
- A submenu does not display its title.

In effect, a submenu acts as an extension to the menu above it. This enables you to display two active menus at the same time, such as if you want the user to choose two options from two exclusive groups of values.

Making a Menu a Submenu

To make a Main Menu a Submenu:

1. Display the menu above the submenu, using `ProMenuCreate()`.
2. Display the submenu, using `ProMenuCreate()`, but make the first argument `PROMENUTYPE_SUB` instead of `PROMENUTYPE_MAIN`.
3. Call `ProMenuProcess()` for the submenu only. Because it is a submenu, the menu above it will become active at the same time.
4. Close both menus, using either `ProMenuDelete()` or `ProMenuDeleteWithStatus()`.

Manipulating Menus

Function Introduced:

- **ProMenubuttonLocationSet()**

The function `ProMenubuttonLocationSet()` provides the ability to move a Creo Parametric menu button to a different location on its menu, or to add new menu buttons to a Creo Parametric menu somewhere other than at the bottom of the menu.

Before you call ProMenubuttonLocationSet(), you must make sure the menu you are modifying has been fully loaded into memory. Make sure ProMenuFileRegister() has been called, and, where appropriate, ProMenuAuxfileRegister().

The first two arguments of the ProMenubuttonLocationSet() function identify the menu and the button, as in ProMenubuttonActionSet().

The final argument is a switch that specifies where to move the button. The possible values are as follows:

- 0—The button becomes the first in the menu.
- 1—The button is inserted after the current first button.
- 2—The button is inserted after the current second button.
- -1—The button becomes the last button on the menu.

Data Menus

Functions Introduced:

- **ProMenuModeSet()**
- **ProMenuDatamodeSet()**

Menus can operate in two modes:

- PROMENUMODE_OPERATIONAL—The default mode. This mode is used in all the previous examples. On an operational menu, only one button is ever set (that is, displayed with a red background) while that command is in progress.
- PROMENUMODE_DATA—Each button remains set until you select it again. This is useful when the buttons do not represent commands, but, for example, a set of independently selectable options.

The function ProMenuModeSet() sets the menu mode. For a PROMENUMODE_DATA menu, you can choose to indicate the set buttons with a check mark instead of the usual red background by using the function ProMenuDatamodeSet().

Calling ProMenuModeSet() and ProMenuDatamodeSet()

The function ProMenuModeSet() has two arguments:

- The menu title
- The menu mode (either PROMENU_MODE_OPERATIONAL or PROMENUMODE_DATA)

The function ProMenuDatamodeSet() has two arguments:

- The menu title.
- The set indicator, which indicates which buttons are set. This argument can have either of the following values:
  - `TRUE`—Use a check mark.
  - `FALSE`—Use a red background. This is the default value.

Both of these functions must be called after the menu has been loaded into memory (using `ProMenuFileRegister()`), and before the menu has been displayed (using `ProMenuCreate()`).

If you want to create a menu whose buttons are dependent on run-time data, use the function `ProMenuStringsSelect()`, described later in this sectionchapter.

Setting Menu Buttons

Functions Introduced:

- **ProMenubuttonHighlight()**
- **ProMenubuttonUnhighlight()**

Sometimes it is useful to be able to set and unset menu buttons from the Creo Parametric TOOLKIT application. For example, if you are using data menus, you can set the appropriate buttons when the menu is displayed to show the current options.

Calling ProMenubuttonHighlight() and ProMenubuttonUnhighlight()

Both `ProMenubuttonHighlight()` and `ProMenubuttonUnhighlight()` take two arguments—the menu title and button name. Both functions must be called after the menu has been displayed (using `ProMenuCreate()`), but before making the menu interactive (using `ProMenuProcess()`). Contrast these rules to the rules for using `ProMenuModeSet()` and `ProMenuDatamodeSet()`.

Controlling Accessibility of Menu Buttons

Functions Introduced:

- **ProMenubuttonActivate()**
- **ProMenubuttonDeactivate()**

A menu button that is inaccessible is one that, though currently displayed on a menu, is gray and has no effect when it is selected. Creo Parametric uses this facility for options that are temporarily unavailable for some reason. For example, you cannot create a hole until you have created the first protrusion.

You can control the accessibility of your own menu buttons from Creo Parametric TOOLKIT using `ProMenubuttonActivate()` and `ProMenubuttonDeactivate()`. Each function takes two arguments: the menu title and button name. These functions must be called when the menu is displayed (after calling `ProMenuCreate()`).

Pushing and Popping Menus

Functions Introduced:

- **ProMenuVisibilityGet()**
- **ProMenuPush()**
- **ProMenuPop()**

Sometimes Creo Parametric temporarily hides certain menus, even though they are still in context, to make room for lower-level menus. An example of this is when you select **Make Datum** during feature creation. This process is called pushing menus, because they are put on a stack from which they can be popped to make them reappear.

The function `ProMenuVisibilityGet()` tells you whether the specified menu is currently displayed. It takes one input argument—the menu title.

The function `ProMenuPush()` pushes the current lowest menu. It takes no arguments.

The function `ProMenuPop()` pops the menu from the top of the stack. It takes no arguments.

Run-time Menus

Functions Introduced:

- **ProMenuStringsSelect()**
- **ProMenuFromStringsRegister()**

The `ProMenuStringsSelect()` function enables you to set up a menu at run time. You do not need to supply a menu file because the buttons are defined when you display the menu. You cannot attach command functions to the button; a run-time menu simply returns a list of the buttons selected.

A run-time menu is displayed together with a submenu that contains the following buttons:

- **Done Select**
- **Quit Select**
- **List**

The default option, **List**, causes the string menu itself to be displayed.

You can set the maximum number of items you want to be selectable. The function returns when the user has selected the maximum number of items you specified, or has selected **Done** or **Quit**. Creo Parametric uses this type of menu to select a disk file to be retrieved after the user selects **Search/Retr**.

The maximum size of the string you assign to a button is `PRO_NAME_SIZE - 1`. `PRO_NAME_SIZE` is defined in file `ProSizeConst.h`.

The function `ProMenuFromStringsRegister()` creates menus at run time and attaches actions to the menu buttons. The function takes as arguments all the information required to create auxiliary (`*.aux`) and user-defined (`*.mnu`) menu files. The first argument is the default menu name. The next argument enables you to specify an alternate name for the menu if, for instance, your application supports a foreign language. The list of button labels is passed to the function as an array of wide character strings. As with the menu name, you can provide alternate button labels for foreign language support. You can also provide one-line Help for each button.

After you have registered the menu with a call to the function `ProMenuFromStringsRegister()`, you can attach actions to the buttons by calling the function `ProMenuButtonActionSet()` for each button. You must also define an exit action for your run-time menu. To do this, call `ProMenuButtonActionSet()` and supply the name of the menu instead of a button name. Finally, create the menu by calling `ProMenuProcess()`, and then `ProMenuCreate()`.

Customizing the Creo Parametric Navigation Area

The Creo Parametric navigation area includes the **Model Tree** and **Layer Tree** pane, **Folder Browser** pane, and **Favorites** pane. The functions described in this section enable Creo Parametric TOOLKIT applications to add custom panes to the Creo Parametric navigation area. The custom panes can contain custom dialog box components or WEB pages.

Adding Custom Web Pages

To add custom web pages to the navigation area, the Creo Parametric TOOLKIT application must:

1. Add a new pane to the navigation area.
2. Set an icon for this pane.
3. Set the URL of the location that will be displayed in the pane.

Functions Introduced:

- **ProNavigatorpaneBrowserAdd()**
- **ProNavigatorpaneBrowsericonSet()**
- **ProNavigatorpaneBrowserURLSet()**

The function `ProNavigatorpaneBrowserAdd()` adds a new pane that can display a web page to the navigation area. The input arguments are:

- *pane_name*—Specify a unique name for the pane. Use this name in susbsequent calls to `ProNavigatorpaneBrowsericonSet()` and `ProNavigatorpaneBrowserURLSet()`.
- *icon_file_name*—Specify the name of the icon file, including the extension. A valid format for the icon file is `.GIF`, `.JPG`, or `.PNG`. The new pane is displayed with the icon image. If you specify the value as `NULL`, the default Creo Parametric icon is used.
- *url*—Specify the URL of the location to be accessed from the pane.

Use the function `ProNavigatorpaneBrowsericonSet()` to set or change the icon of a specified browser pane in the navigation area.

Use the `ProNavigatorpaneBrowserURLSet()` to change the URL of the page displayed in the browser pane in the navigation area.

Adding Custom Dialog Box Components

To add a new pane to the navigation area based on Creo Parametric TOOLKIT dialog box components:

1. Add a new pane to the navigation area.
2. Assign an icon for the pane.
3. Add the Creo Parametric TOOLKIT dialog box components using one of the following methods:
    - Assign resource files that describe the overall structure of the new navigation pane. To customize the components after placement in the navigation pane, get the window device name and component name using the Creo Parametric TOOLKIT functions described in this section and set

the necessary values using the Creo Parametric TOOLKIT functions described in the chapter User Interface: Dialogs on page 344

- To build the layout of the dialog box programmatically, obtain the window device name and layout name. Add each component to the layout as desired using Creo Parametric TOOLKIT user interface functions described in the chapter User Interface: Dialogs on page 344.

Functions Introduced:

- **ProNavigatorpanePHolderAdd()**
- **ProNavigatorpanePHolderDelete()**
- **ProNavigatorpanePHolderShow()**
- **ProNavigatorpanePHolderHide()**
- **ProNavigatorpanePHolderDevicenameGet()**
- **ProNavigatorpanePHolderLayoutGet()**
- **ProNavigatorpanePHolderComponentnameGet()**
- **ProNavigatorpanePHolderHelptextSet()**

The function `ProNavigatorpanePHolderAdd()` adds a layout that will be displayed in the new pane in the navigation area. The input arguments are:

- *pane_name*— Specify a unique name for the pane.
- *resource_name*—Specify the name of the resource file to use (whose top component must be a layout, not a dialog box). The contents of the layout from the specified resource file will be inserted into the custom pane.
- *icon_file_name*—Specify the name of the icon file, including the extension. A valid format for the icon file is `.GIF`, `.JPG`, or `.PNG`. The new pane is displayed with the icon image. If you specify the value as `NULL`, the default Creo Parametric icon is used.

The function `ProNavigatorpanePHolderDelete()` deletes the specified pane from the navigation area.

Use the functions `ProNavigatorpanePHolderShow()` and `ProNavigatorpanePHolderHide()` to show and hide the specified pane in the navigation area.

The function `ProNavigatorpanePHolderLayoutGet()` returns the layout name for the specified pane in the navigation area. You can create and place the user interface components within this layout.

Components added to the custom pane actually belong to the Creo Parametric main window dialog. Creo Parametric automatically modifies the names of components loaded from resource files to ensure that no name collisions occur when the components are added. The functions

ProNavigatorpanePHolderDevicenameGet() and ProNavigatorpanePHolderComponentnameGet() allow you to locate the names of components that you need to access.

Use the function ProNavigatorpanePHolderDevicenameGet() to obtain name of the Creo Parametric window owning the new pane in the navigation area.

Use the function ProNavigatorpanePHolderComponentnameGet() to obtain the complete name of the component in the navigation pane, if loaded from a layout.

Use the device name and the component name to add or update the placement of the components in the layout with the help of the ProUI* functions. For more information on the user interface functions, refer to the chapter User Interface: Dialogs on page 344.

The function ProNavigatorpanePHolderHelptextSet() sets the popup help text, which is displayed when you point over the navigator pane component.

Example 11: Customizing the Creo Parametric Navigation Pane

The sample code in the file UgNavigatorPane.c located at <creo_toolkit_loadpoint>protk_appls/pt_userguide/ptu_menu shows you how to customize the Creo Parametric navigation pane.

Registering Notifications to Add and Destroy Content to a New Pane

The navigation panes are available in every window within a Creo Parametric session. Notifications are provided to the Creo Parametric TOOLKIT application when a Creo Parametric window populated with a model so that the application can add the necessary contents to the new pane upon this event. Similarly, notifications are provided before a model is removed from a Creo Parametric window so that the application can cleanup resources related to added panes. For more information on using notifications see the chapter Event-driven Programming: Notifications on page 2010.

Functions Introduced:

- **ProNotificationSet()**
- **ProWindowOccupyPostAction()**
- **ProWindowVacatePreAction()**

Specify the argument *type* of the function ProNotificationSet() to PRO_WINDOW_OCCUPY_POST, to call the callback function whose signature matches ProWindowOccupyPostAction(). This function is called when a new Creo

Parametric window is created, or when the base window is populated and is used by the application to add the necessary content to the new pane in the navigation area.

Specify the argument *type* of the function `ProNotificationSet()` to `PRO_WINDOW_VACATE_PRE`, to call the callback function whose signature matches `ProWindowVacatePreAction()`. This function is called when a Creo Parametric window is closed, or when the base window gets cleared. Use this function to free memory allocated by the Creo Parametric TOOLKIT application to add content in the navigation pane.

Entering Creo Parametric Commands

Functions Introduced:

- **ProMacroLoad()**
- **ProMacroExecute()**
- **ProMenuCommandPush()**

The function `ProMacroLoad()` loads a macro string or a mapkey onto a stack of macros that are executed after control returns to Creo Parametric. A Creo Parametric TOOLKIT macro is a string. The macro string is equivalent to a mapkey without the key sequence and the mapkey name.

Note

`ProMacroLoad()` fails if a macro string contains a backslash or if a command splits over two lines with or without a backslash. A macro string can contain multiple commands separated by semicolons. However, each command should entirely appear in a single line.

The function `ProMacroLoad()` enables you to execute the commands created by the Creo Parametric TOOLKIT application using the following macro:

```
~ Command `<command name>`
```

Click **Mapkeys Settings** in the **Environment** tab of the **Creo Parametric Options** dialog box to create a mapkey in the Creo Parametric user interface. Copy the value of the generated mapkey option from the **Configuration Editor** in the **Creo Parametric Options** dialog box. An example of a created mapkey is as follows:

```
$F2 @MAPKEY_LABELtest;
~ Command `ProCmdModelNew`
~ Activate `new` `OK`
```

The key sequence is `$F2.` . The remainder of the string after the first semicolon is the macro string. In this case, it is as follows:

```
~ Command `ProCmdModelNew`
```

```
~ Activate `new` `OK`
```

You can either pass the mapkey directly or the generated macro string to `ProMacroLoad()`. Pass the mapkey directly as *%key_sequence*.

Note

Creating or editing the macro string manually is not supported, as mapkeys are not a supported scripting language. The syntax is not defined for users and may not remain constant across different datecodes of Creo Parametric.

Execution Rules

Consider the following rules about the execution of macros:

- In asynchronous mode, macros are executed as soon as they are loaded with `ProMacroLoad()`.
- In synchronous mode, the mapkey or the macro strings are pushed onto a stack and are popped off and executed only when control returns to Creo Parametric from the Creo Parametric TOOLKIT program. Due to the last in, first out nature of the stack, macros that cannot be passed entirely in one `ProMacroLoad()` call should have the strings loaded in reverse order of desired execution.
- To execute a macro from within Creo Parametric TOOLKIT, call the function `ProMacroExecute()`. The function runs the Creo Parametric macro and returns the control to the Creo Parametric TOOLKIT application. It executes the macros previously loaded using the function `ProMacroLoad()`. The function works only in the synchronous mode.
- Do not call the function `ProMacroExecute()` during the following operations:
  - Activating dialog boxes or setting the current model
  - Erasing the current model
  - Replaying a trail file
- Clicking the **OK** button on the dialog box to complete the command operation. In this case, the dialog box may be displayed momentarily without completing the command operation.

Note

- You can execute only the dialog boxes with built-in exit confirmation as macros, by canceling the exit action.
- It is possible that a macro may not be executed because a command specified in the macro is currently inaccessible in the menus. The functional success of `ProMacroExecute()` depends on the priority of the executed command against the current context.

- If some of the commands ask for an input to be entered from the keyboard (such as a part name), the macro continues execution after you type the input and press `ENTER`. However, if you must select something with the mouse (such as selecting a sketching plane), the macro is interrupted and ignores the remaining commands in the string.

  This allows the application to create object-independent macros for long sequences of repeating choices (as long as the user does not have to select any geometry).

A `ProStringToWstring()` call for defining the macro string must be followed by the following calls:

- `ProMacroLoad(macro wstring)` to load the macro strings or the mapkey
- `ProMacroExecute()` to execute the macro

Some sample macros in various scenarios are given below.

Ribbon User Interface Macros

The following single entry and exit type of interactions are supported by `ProMacroExecute()`.

- To switch the wireframe display of model, use the macro:
  ```
  Command `ProCmdEnvWireframe` 1
  ```
- To switch the shaded display of model, use the macro:
  ```
  ~ Command `ProCmdEnvShaded` 1
  ```
- You can switch the display for datum planes and datum axes using the following macros:
  - Datum Planes
    ```
    ~ Command `ProCmdEnvDtmDisp` 1
    ```
  - Datum Axes
    ```
    ~ Command `ProCmdEnvAxisDisp` 0
    ```

- To repaint a model, use the macro:
  ```
  ~ Command `ProCmdViewRepaint`
  ```
- To get the default model orientation, use the macro:
  ```
  ~ Command `ProCmdNamedViewsGalSelect` `Default`
  ```
- To get the model information, use the macro:
  ```
  ~ Command `ProCmdInfoModel`
  ```

Macros For Feature Creation

The following macros are used while creating the following features.

- To create a hole feature, use the macro:
  ```
  ~ Command `ProCmdHole`
  ```
- To extrude a feature, use the macro:
  ```
  ~ Command `ProCmdFtExtrude`
  ```
- To create a datum plane, use the macro:
  ```
  ~ Command `ProCmdDatumPlane`
  ```

Creo Parametric Navigator Macros

The following macros are provided for Creo Parametric navigator:

- For folder navigator:
  ```
  ~ Select `main_dlg_cur` `PHTLeft.ProExplorerTab` 1 `PHTLeft.Folders`
  ```
- For Favorites navigator:
  ```
  ~ Select `main_dlg_cur` `PHTLeft.ProExplorerTab` 1 `PHTLeft.FavLay`
  ```
- For the Model Tree:
  ```
  ~ Select `main_dlg_cur` `PHTLeft.ProExplorerTab` 1 `PHTLeft.MdlTreeLay`
  ```

The function `ProMenuCommandPush()` places the name of a specific menu button in the command input buffer for Creo Parametric. This command is executed after control returns to Creo Parametric from the Creo Parametric TOOLKIT application, as if you have selected that menu button. This menu button must be from the menu that is currently displayed in the Creo Parametric user interface.

Specifying Keyboard Input

You can specify keyboard input within the command string. As previously specified, a macro must be preceded by a pound sign (#) and terminated by a semicolon. If the field after the semicolon does not start with a pound sign, the data up to the next semicolon is used as input at the next keyboard prompt. If the command currently being executed does not request keyboard input, the system ignores this keyboard data. Note that keyboard data is case-sensitive and spaces are not ignored. A carriage return is indicated when no data appears between the semicolons.

Note

Note that the correctness of the sequence is the responsibility of the user. PTC does not guarantee that a sequence will be valid from one version of Creo Parametric to another.

18

User Interface: Dialogs

This chapter describes the User Interface (UI) components available in Pro/TOOLKIT for Pro/ENGINEER Wildfire 3.0 onwards. The following sections introduce each of the dialog component types, operations and callbacks available for each component, and the methods and techniques that can be used to instantiate and show customized user interface dialogs.

Introduction

This chapter includes documentation for each UI component. The documentation is divided into the following sections:

- Attributes—Defines the names and functions that affect attributes on the UI component. Each component supports its own set of unique attributes; however, some attributes are supported by more than one component. Because of the fact that attributes typically work on more than one component, detailed documentation for the attributes is included in a master table at the end of this chapter.
- Operations—Defines the component-specific functions that make more detailed modifications to the components.
- Actions—Defines the functions that register action callbacks on a component.

Note

From Creo 3.0 onward, a new tool, Creo UI Editor, allows you to interactively create and edit dialog boxes for Creo Object TOOLKIT C++ and Creo Object TOOLKIT Java customizations. It provides a library of graphical user interface components such as buttons, lists, and so on. The new framework, User Interface Foundation Classes (UIFC), provides enhanced attributes and actions for the user interface components. The UIFC framework is available in Creo Object TOOLKIT C++ and Creo Object TOOLKIT Java. You can generate callbacks in Creo Object TOOLKIT C++ or Creo Object TOOLKIT Java. Refer to the *Creo UI Editor User's Guide*, for more information.

About Creo Parametric TOOLKIT Support for User Interface

Creo Parametric TOOLKIT allows applications to create dialogs and dashboards with the same look and feel as those in Creo Parametric. Creo Parametric TOOLKIT users can accomplish this task by following the following steps:

- Establish the main UI container object. This is either a dialog or a dashboard. Optionally, this can be read from a resource file to prepopulate the container with specific components.
- Add components to the container (if they do not already exist).
- Set attributes on components in the container.

- Execute operations on components in the container. Operations also modify the component, but typically make more detailed or sophisticated changes than setting an individual attribute.
- Establish action function callbacks on components in the container. Action functions are called when the user interacts with the component in some way. They allow the application to react to user events.
- Show the dialog or dashboard.
- Based on user actions, eventually you will need to close the container.
- "Destroy" the container to free up the resources it uses.

> **Note**
>
> The functions described in this section do not support using Creo Parametric TOOLKIT to modify standard Creo Parametric dialogs.

The UI function library is integral to Creo Parametric, not the Creo Parametric TOOLKIT libraries; thus, the library can only be used while a Creo Parametric session is active. The library cannot be used to display user interfaces when the application is not connected to Creo Parametric running interactively.

UI Components

The behavior and uses of the different component types is introduced briefly below, and described in more detail in later sections. The component types are:

- Tab—part of a dialog that can contain several groups of components, formatted such that only one group is visible at a time. A Tab component must always contain a set of Layout components; each layout contains the components that must displayed at one time. The Figure - ‘All Components Dialog’ shows a decorated Tab which displays a handle on each layout to allow the user to select which layout is visible.
- Layout—an area of a dialog which can contain any number of other dialog components. A Layout can be used to better control the relative position of components in a dialog, by allowing the grids in different parts of the dialog to adopt unaligned rows or columns. A layout can also be used inside a Tab component.
- Check Button—a button which toggles between a `TRUE` and `FALSE` state each time the user selects it.
- Drawing Area—a component which allows points, lines, shapes, images and text (including symbols) to be drawn in a variety of colors.

- Input Panel—a box containing a single line of text. The Input Panel may be set to expect text in different formats, for example a real number or an integer. The Input Panel may also be set to be read-only, when it is used by the application to show information to the user.
- Label—a text string used to label the other components.
- List—a box containing a list of text strings, which can be selected by the user. Users can set the List to allow selection of only one item at a time, or more than one.
- Option Menu—a single-line box which allows selection of a single text string from a list of options. The selection is done using a pull-down menu, which appears when a button next to the text box is selected.
- Progress Bar—a component which shows the progress of a time-consuming action.
- Push Button—a button which performs some action when it is selected. It does not contain any remembered state. Push Buttons appear on almost every dialog as `OK` and `Cancel` buttons.
- Radio Group—a set of buttons which individually act like check buttons, but which are connected to each other such that only one can be set to `TRUE` at any time. Selecting one button sets that button and unsets all others in the group.
- Separator—a separator is for cosmetic purposes only, and helps to visually divide components into logical groups.
- Slider—a device which allows the user to set a value in a predefined range by moving a handle with the mouse. Use sliders in situations where an exact value may not be needed. A slider should usually be tied programmatically with a read-only input panel to show the current value.
- Spin-Box—a box containing a single numerical value that can be directly edited. The spin box also has up- and down-arrow buttons for increasing or decreasing the value in steps. A single click increments or decrements by a single step. Holding a button down makes the value change in repeated steps, first small steps and then large steps. The step sizes can be set for each spin box.
- Table—a set of tabulated rows and columns containing text and other components.
- Text Area—a box containing unformatted text containing any number of lines. It may be set to be read-only and used by the application to output information to the user.

- Thumbwheel—a thumbwheel is similar to slider but provides finer control over a wider range of values. Unlike the slider, it does not provide a visual indication of the current value.
- Tree—a tree contains nodes which are structured and displayed in a hierarchical formation.

Naming Convention for UI Components

The valid characters for naming UI components are:

- A to Z (uppercase)
- a to z (lowercase)
- 0 to 9
- Underscore(_)
- Hypen(-)

Using any other characters in UI component names may result in an error.

Menubars and Menubar Components

A dialog can also contain its own menubar. These menubars support cascading menus.

See Example 15: Resource File for Dialog with Menubar on page 457 for resource file code for this example.

The following components are used to define menu bars and their dependent menus:

- Menubar—The menubar itself is just a container for the menu panes. A dialog can contain only one menubar, and it must contain at least one other component at the top level.
- MenuPane—A menu pane describes a button on a menubar and also acts as a container for the components on the pull-down menu that appears when the user selects the menu pane button.
- Cascade button—A button on a pull-down menu that contains its own menupane. Selecting the cascade button pulls out the menu described by the menupane.

The following components described in the previous section can also be added to menu panes, but in this case their appearance is automatically modified to suit the style of pull-down menus:

- Check Button—This looks like a regular menu button, but in fact toggles its state. When `TRUE`, it shows a check mark next to the label.
- Push Button—When added to a menu pane a pushbutton represents a command that causes some action.
- Radio Group—A radio group on a menu pane behaves exactly as it would in the body of a dialog, although the appearance is rather different, as shown in the picture above.
- Separator—A separator can be used to group buttons on a menu pane.

Dialog Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUIDialogIsAttachedBottom()	ProUIDialogAttachBottom() ProUIDialogUnattachBottom()
.AttachTop on page	ProUIDialogIsAttachedTop()	ProUIDialogAttachTop() ProUIDialogUnattachTop()
.Attach Right on page	ProUIDialogIsAttachedRight()	ProUIDialogAttachRight() ProUIDialogUnattachRight()
.AttachLeft on page	ProUIDialogIsAttachedLeft()	ProUIDialogAttachLeft() ProUIDialogUnattachLeft()
.BottomOffset on page	ProUIDialogBottomoffsetGet()	ProUIDialogBottomoffsetSet()
.Bitmap on page	ProUIDialogBitmapGet()	ProUIDialogBitmapSet()
.ChildNames on page	ProUIDialogChildnamesGet()	Not Applicable
.ClassName on page	ProUIComponentClassnameGet()	Not Applicable
.DefaultButton on page	ProUIDialogDefaultbuttonGet()	ProUIDialogDefaultbuttonSet()
.Dialog Style on page	ProUIDialogDialogstyleGet()	ProUIDialogDialogstyleSet()
.Focus on page	Not Applicable	ProUIDialogFocusSet()
.Height on page	ProUIDialogHeightGet()	ProUIDialogHeightSet()
.HorzAt Point on page	ProUIDialogHorzatpointGet()	ProUIDialogHorzatpointSet()
.HorzDialog on page	ProUIDialogHorzdialogGet()	ProUIDialogHorzdialogSet()
.HorzMethod on page	ProUIDialogHorzmethodGet()	ProUIDialogHorzmethodSet()

Attribute Name	Get Function	Set Function(s)
.HorzPoint on page	ProUIDialogHorzpointGet()	ProUIDialogHorzpointSet()
.HorzPosOffset on page	ProUIDialogHorzposoffsetGet()	ProUIDialogHorzposoffsetSet()
.HorzSize on page	ProUIDialogHorzsizeGet()	ProUIDialogHorzsizeSet()
.Labels on page	ProUIDialogTitleGet()	ProUIDialogTitleSet()
.LeftOffset on page	ProUIDialogLeftoffsetGet()	ProUIDialogLeftoffsetSet()
.Mapped on page	ProUIDialogIsMapped()	ProUIDialogMappedSet() ProUIDialogMappedUnset()
.PopupMenu on page	ProUIDialogPopupmenuGet()	ProUIDialogPopupmenuSet()
.RememberPosition on page	ProUIDialogRemembersPosition()	ProUIDialogRememberPosition() ProUIDialogForgetPosition()
.RememberSize on page	ProUIDialogRemembersSize()	ProUIDialogRememberSize() ProUIDialogForgetSize()
.Resizeable on page	ProUIDialogIsResizeable()	ProUIDialogEnableResizing() ProUIDialogDisableResizing()
.RightOffset on page	ProUIDialogRightoffsetGet()	ProUIDialogRightoffsetSet()
.StartLocation on page	ProUIDialogStartlocationGet()	ProUIDialogStartlocationSet()
.TopOffset on page	ProUIDialogTopoffsetGet()	ProUIDialogTopoffsetSet()
.VertAtPoint on page	ProUIDialogVertatpointGet()	ProUIDialogVertatpointSet()
.VertDialog on page	ProUIDialogVertdialogGet()	ProUIDialogVertdialogSet()
.VertMethod on page	ProUIDialogVertmethodGet()	ProUIDialogVertmethodSet()
.VertPoint on page	ProUIDialogVertpointGet()	ProUIDialogVertpointSet()
.VertPosOffset on page	ProUIDialogVertposoffsetGet()	ProUIDialogVertposoffsetSet()
.VertSize on page	ProUIDialogVertsizeGet()	ProUIDialogVertsizeSet()
.Width on page	ProUIDialogWidthGet()	ProUIDialogWidthSet()

Dialog Operations

Functions Introduced

- **ProUIDialogCreate()**
- **ProUIDialogActivate()**
- **ProUIDialogComponentsCollect()**
- **ProUIDialogMinimumsizeGet()**
- **ProUIDialogSizeGet()**
- **ProUIDialogPositionReset()**
- **ProUIDialogReconfigure()**
- **ProUIDialogScreenpositionGet()**
- **ProUIDialogAboveactivewindowGet()**
- **ProUIDialogAboveactivewindowSet()**
- **ProUIDialogShow()**
- **ProUIDialogHide()**
- **ProUIDialogExit()**
- **ProUIDialogDestroy()**
- **ProUITimerCreate()**
- **ProUIDialogTimerStart()**
- **ProUIDialogTimerStop()**
- **ProUITimerDestroy()**

The function `ProUIDialogCreate()` loads a dialog from a resource file into memory. It can also create an empty dialog by passing a `NULL` value for the resource file name. The input arguments follow:

- *session_dialog_name*—Name of the dialog.
- *resource*—Name of the resource file.

Note

- The function `ProUIDialogCreate()` requires that the resource file name input matches the case of the actual resource file.
- The dialog name specified in the resource file should match both the input argument *resource* and the name of the resource file without adding the suffix name.

The following points must be noted while developing Creo Parametric TOOLKIT applications:

- Resource names of dialogs resources must not coincide with the resources that are defined by PTC.
- Dialog instance or session names must not coincide with the instance or session names that are defined by PTC.
- You can ensure uniqueness in the dialog resource and session names by adding a prefix that specifies the name of the Creo Parametric TOOLKIT application.

Use the function `ProUIDialogActivate()` to display the dialog on the screen and makes it active by setting the key input focus on it. This function returns only after the function `ProUIDialogExit()` has been called on the same dialog.

Use the function `ProUIDialogComponentsCollect()` to return the names of components in this dialog. This function can also filter components by their type.

Note

Refer to the section UI Components on page 347 for the predefined list of component types.

Use the function `ProUIDialogMinimumsizeGet()` to get the minimum size of the dialog in pixels.

Use the function `ProUIDialogSizeGet()` to get the size of the dialog.

Use the function `ProUIDialogPositionReset()` to reset the dialog in its previous screen position.

Use the function `ProUIDialogReconfigure()` to reset the size and position of the dialog.

Use the function `ProUIDialogScreenpositionGet()` get the screen position in pixels of the dialog component.

Use the function `ProUIDialogAboveactivewindowGet()` to checks if the dialog is always going to be above the Creo Parametric dialogs currently active in the Creo Parametric window.

Use the function `ProUIDialogAboveactivewindowSet()` to set focus of the dialog above any dialog in the currently active Creo Parametric window.

Note

Using the `ProUIDialogAboveactivewindowSet()` allows Creo Parametric TOOLKIT applications to always stay in focus in the Creo Parametric window during opening and closing events of Creo Parametric.

Use the `ProUIDialogDestroy()` to remove the dialog instance from memory as it is not automatically removed.

Use the function `ProUIDialogShow()` to show a secondary window when the primary window is being restored.

Use the function `ProUIDialogHide()` to iconify a secondary window when the primary window is being minimized.

Use the function `ProUIDialogExit()` to terminate the activation of the named dialog window. The function causes a return from the call to `ProUIDialogActivate()` that make it active.

Use the function `ProUIDialogDestroy()` to remove the dialog from the memory.

You can set a timer to schedule an action to be executed later in Creo Parametric session. The function `ProUITimerCreate()` creates a timer and registers the action. f you call the function `ProUITimerCreate()` with `timer_name` that already exists in the session, it returns an error. The input arguments for the function are:

- *action*—Specifies call to the action callback function `ProUITimerAction()`. You can invoke the action callback only once from the function `ProUIDialogTimerStart()`. To run the action again, you must call `ProUIDialogTimerStart()` again.
- *appdata*—Specifies the data that will be passed to the action callback.
- *timer_name*—Specifies the name of the timer.

The function returns the ID of the timer, which will be used to start and stop the timer.

Use the function `ProUIDialogTimerStart()` to start the timer. The action is executed after the specified interval of time. The input arguments are:

- *dialog*—Specifies the name of the dialog box. You can invoke only one timer at a time for a dialog box.
- *timer_id*—Specifies the ID of the timer, which was returned by the function `ProUITimerCreate()`.

- *duration*—Specifies the time interval in milliseconds after which the action must be executed. The minimum value is set to 500 milliseconds.
- *write_in_trail_file*—Specifies if the timer action must be recorded in a trail file.

The function `ProUIDialogTimerStop()` stops the timer and the action will not be executed. Specify the ID of the timer as the input argument.

The function `ProUITimerDestroy()` removes the specified timer, which was created using the function `ProUITimerCreate()`. If the timer to be removed has already been started, it is first stopped, and then removed. Pass the ID of the timer as the input argument. After the timer is removed, its name is available for use again. You can specify the name to a new timer.

Note

You must use the functions `ProUITimerCreate()`, `ProUIDialogTimerStart()`, `ProUIDialogTimerStop()`, and `ProUITimerDestroy()` only in DLL mode.

Example 1: Source for Dialog with Text Question, OK and Cancel Buttons

The example in the file `UgUIYesnoDialog.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui`, shows the source code that uses this dialog.

Adding and Removing Components

Component Name	Add Function	Remove Function
`Checkbutton`	`ProUIDialogCheckbuttonAdd()`	`ProUIDialogCheckbuttonRemove()`
`Drawingarea`	`ProUIDialogDrawingareaAdd()`	`ProUIDialogDrawingareaRemove()`
`Inputpanel`	`ProUIDialogInputpanelAdd()`	`ProUIDialogInputpanelRemove()`
`Label`	`ProUIDialogLabelAdd()`	`ProUIDialogLabelRemove()`
`Layout`	`ProUIDialogLayoutAdd()`	`ProUIDialogLayoutRemove()`
`List`	`ProUIDialogListAdd()`	`ProUIDialogListRemove()`
`Menubar`	`ProUIDialogMenubarAdd()`	`Not Applicable`
`Menupane`	`ProUIDialogMenupaneAdd()`	`ProUIDialogMenupaneRemove()`
`Optionmenu`	`ProUIDialogProgressbarAdd()`	`ProUIDialogOptionmenuRemove()`
`Progressbar`	`ProUIDialogProgressbarAdd()`	`ProUIDialogProgressbarRemove()`
`Pushbutton`	`ProUIDialogPushbuttonAdd()`	`ProUIDialogPushbuttonRemove()`

Component Name	Add Function	Remove Function
Radiogroup	ProUIDialogRadiogroupAdd()	ProUIDialogRadiogroupRemove()
Separator	ProUIDialogSeparatorAdd()	ProUIDialogSeparatorRemove()
Slider	ProUIDialogSliderAdd()	ProUIDialogSliderRemove()
Spinbox	ProUIDialogSpinboxAdd()	ProUIDialogSpinboxRemove()
Tab	ProUIDialogTabAdd()	ProUIDialogTabRemove()
Table	ProUIDialogTableAdd()	ProUIDialogTableRemove()
Textarea	ProUIDialogTextareaAdd()	ProUIDialogTextareaRemove()
Thumbwheel	ProUIDialogThumbwheelAdd()	ProUIDialogThumbwheelRemove()
Tree	ProUIDialogTreeAdd()	ProUIDialogTreeRemove()

Creo Parametric TOOLKIT provides functions to add components to a dialog. These functions accept an argument of type `ProUIGridopts` that determines the location and initial placement details of the component. In addition, a number of grid-specific attributes control the position and resizing of the newly created component within the grid of the Dialog.

These grid-specific attributes are listed as follows:

Attribute	Default Value	Description
column	PRO_UI_INSERT_NEW_COLUMN	The column of the grid into which the component should be added. A value of PRO_UI_INSERT_NEW_COLUMN indicates that the component should be added to a newly created column to the left of any existing columns.
row	PRO_UI_INSERT_NEW_ROW	The row of the grid into which the component should be added. A value of PRO_UI_INSERT_NEW_ROW indicates that the component should be added to a newly created row to the left of any existing rows.
horz_cells	1	The number of cells which the component should occupy in the existing grid in a horizontal direction.
vert_cells	1	The number of cells which the component should occupy in the existing grid in a vertical direction.
horz_resize	PRO_B_TRUE	A flag indicating whether the grid cell containing the component should resize horizontally.
vert_resize	PRO_B_TRUE	A flag indicating whether the grid cell containing the component should resize vertically.
attach_top	PRO_B_TRUE	Attach the item to the top neighbor
attach_bottom	PRO_B_TRUE	Attach the item to the bottom neighbor

Attribute	Default Value	Description
attach_left	PRO_B_TRUE	Attach the item to the left neighbor
attach_right	PRO_B_TRUE	Attach the item to the right neighbor
top_offset	PRO_UI_USE_DEVICE_OFFSET	Offset to the top neighbor. The default value PRO_UI_USE_DEVICE_OFFSET inherits the offset from the owning dialog.
bottom_offset	PRO_UI_USE_DEVICE_OFFSET	Offset to the bottom neighbor. The default value PRO_UI_USE_DEVICE_OFFSET inherits the offset from the owning dialog.
left_offset	PRO_UI_USE_DEVICE_OFFSET	Offset to the left neighbor. The default value PRO_UI_USE_DEVICE_OFFSET inherits the offset from the owning dialog.
right_offset	PRO_UI_USE_DEVICE_OFFSET	Offset to the right neighbor. The default value PRO_UI_USE_DEVICE_OFFSET inherits the offset from the owning dialog.

Note

Components that are added to a dialog after it is displayed do not permit modification of all component attributes. When creating and displaying a dialog, it is preferable to use these functions to add the components before activating the dialog. If a component might be needed but should not be shown initially, add it before activation and set its .Visible attribute to false.

Dialog Action Callbacks

Functions Introduced

- **ProUIDialogPremanagenotifyActionSet()**
- **ProUIDialogPostmanagenotifyActionSet()**
- **ProUIDialogDestroynotifyActionSet()**
- **ProUIDialogCloseActionSet()**
- **ProUIDialogActivateActionSet()**
- **ProUIDialogAppActionSet()**
- **ProUIDialogAppActionRemove()**

Use the function `ProUIDialogPremanagenotifyActionSet()` to set the function to be called when the dialog is about to be managed. For example, when a dialog box is displayed or redisplayed.

Use the function `ProUIDialogPostmanagenotifyActionSet()` to set the function to be called when the dialog has just been managed. For example, when a dialog box is displayed.

Use the function `ProUIDialogDestroynotifyActionSet()` to set the function to be called when the dialog is about to be destroyed.

Use the function `ProUIDialogCloseActionSet()` to set the action function to be called when the user attempts to close the dialog using the window close icon in the upper right corner of the dialog.

Use the function `ProUIDialogActivateActionSet()` to set the function to be called when the dialog has just been activated and made the current foreground window. The action function for a given dialog can be called

- The dialog must not be the current foreground application.
- The dialog (when it is not the foreground application) is activated using one of the following methods:
  - When the user clicks on the taskbar button for the given dialog.
  - When the user switches to the given dialog using `Alt+Tab`.
  - When the user clicks within the given dialog.

Use the function `ProUIDialogAppActionSet()` to set a function to be called only once, when you return to or enter an event loop. The input arguments follow:

- *dialog*—Name of the dialog. The action is associated with the dialog and is automatically cancelled if the dialog is destroyed. The value can be `NULL`.
- *function*—Function to be called when you return to an event loop.
- *data*—Action data passed to the callback function. The value can be `NULL`.

Use the function `ProUIDialogAppActionRemove()` to remove a function added via `ProUIDialogAppActionSet()`. The input arguments follow:

- *dialog*—Name of the dialog passed to the function `ProUIDialogAppActionRemove()`.
- *function*— Function passed to `ProUIDialogActionSet()`.
- *data*—Action data passed to `ProUIDialogActionSet()`.

Cascade Button

Cascade Button Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUICascadebutton IsAttachedBottom()	ProUICascadebutton AttachBottom() ProUICascadebutton UnattachBottom()
.AttachTop on page	ProUICascadebutton IsAttachedTop()	ProUICascadebutton AttachTop() ProUICascadebutton UnattachTop()
.AttachRight on page	ProUICascadebutton IsAttachedRight()	ProUICascadebutton AttachRight() ProUICascadebutton UnattachRight()
.AttachLeft on page	ProUICascadebutton IsAttachedLeft()	ProUICascadebutton AttachLeft() ProUICascadebutton UnattachLeft()
.BottomOffset on page	ProUICascadebutton BottomoffsetGet()	ProUICascadebutton BottomoffsetSet()
.Bitmap on page	ProUICascadebutton BitmapGet()	ProUICascadebutton BitmapSet()
.CascadeDirection on page	ProUICascadebutton CascadedirectionGet()	ProUICascadebutton CascadedirectionSet()
.ChildNames on page	ProUICascadebutton ChildnamesGet()	ProUICascadebutton ChildnamesSet()
.HelpText on page	ProUICascadebutton HelptextGet()	ProUICascadebutton HelptextSet()
.Label on page	ProUICascadebutton TextGet()	ProUICascadebutton TextSet()
.LeftOffset on page	ProUICascadebutton LeftoffsetGet()	ProUICascadebutton LeftoffsetSet()
.ParentName on page	ProUICascadebutton ParentnameGet()	ProUICascadebutton ParentnameSet()
.PopupMenu on page	ProUICascadebutton PopupmenuGet()	ProUICascadebutton PopupmenuSet()
.Resizeable on page	ProUICascadebutton IsResizeable()	ProUICascadebutton EnableResizing() ProUICascadebutton DisableResizing()
.RightOffset on page	ProUICascadebutton RightoffsetGet()	ProUICascadebutton RightoffsetSet()
.Sensitive on	ProUICascadebutton IsEnabled()	ProUICascadebutton Enable()

Attribute Name	Get Function	Set Function(s)
page		ProUICascadebutton Disable()
.TopOffset on page	ProUICascadebutton TopoffsetGet()	ProUICascadebutton TopoffsetSet()
.Visible on page	ProUICascadebutton IsVisible()	ProUICascadebutton Show() ProUICascadebutton Hide()

Checkbutton

Checkbutton Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUICheckbuttonIsAttachedBottom()	ProUICheckbuttonAttachBottom() ProUICheckbuttonUnattachBottom
.AttachTop on page	ProUICheckbuttonIsAttachedTop()	ProUICheckbuttonAttachTop() ProUICheckbuttonUnattachTop()
.AttachRight on page	ProUICheckbuttonIsAttachedRight()	ProUICheckbuttonAttachRight() ProUICheckbuttonUnattachRight()
.AttachLeft on page	ProUICheckbuttonIsAttachedLeft()	ProUICheckbuttonAttachLeft() ProUICheckbuttonUnattachLeft()
.BottomOffset on page	ProUICheckbuttonBottomoffsetGet()	ProUICheckbuttonBottomoffsetSet()
.Bitmap on page	ProUICheckbuttonBitmapGet()	ProUICheckbuttonBitmapSet()
.ButtonStyle on page	ProUICheckbuttonButtonstyleGet()	ProUICheckbuttonButtonstyleSet()
.HelpText on page	ProUICheckbuttonHelptextGet()	ProUICheckbuttonHelptextSet()
.Label on page	ProUICheckbuttonTextGet()	ProUICheckbuttonTextSet()
.ModalOverride on page	Not Applicable	ProUICheckbuttonModaloverrideSet()
.LeftOffset on page	ProUICheckbuttonLeftoffsetGet()	ProUICheckbuttonLeftoffsetSet()
.ParentName on page	ProUICheckbuttonParentnameGet()	Not Applicable

Attribute Name	Get Function	Set Function(s)
.PopupMenu on page	ProUICheckbuttonPopupmenuGet()	ProUICheckbuttonPopupmenuSet()
.Resizeable on page	ProUICheckbuttonIsResizeable()	ProUICheckbuttonEnableResizing() ProUICheckbuttonDisableResizing()
.RightOffset on page	ProUICheckbuttonRightoffsetGet()	ProUICheckbuttonRightoffsetSet()
.Set on page	ProUICheckbuttonGetState()	ProUICheckbuttonSet() ProUICheckbuttonUnset()
.Sensitive on page	ProUICheckbuttonIsEnabled()	ProUICheckbuttonEnable() ProUICheckbuttonDisable()
.TopOffset on page	ProUICheckbuttonTopoffsetGet()	ProUICheckbuttonTopoffsetSet()
.Visible on page	ProUICheckbuttonIsVisible()	ProUICheckbuttonShow() ProUICheckbuttonHide()

Checkbutton Operations

Functions Introduced

- **ProUICheckbuttonAnchorSet()**
- **ProUICheckbuttonSizeSet()**
- **ProUICheckbuttonMinimumsizeGet()**
- **ProUICheckbuttonPositionSet()**
- **ProUICheckbuttonPositionGet()**
- **ProUICheckbuttonSizeGet()**

Use the function `ProUICheckbuttonAnchorSet()` to set the position of the checkbutton with respect to a given anchor location. This function is applicable only if the parent of the checkbutton is a drawing area. The input argument *anchor* determines which part of the component is being positioned.

Use the function `ProUICheckbuttonSizeSet()` to set the size of the checkbutton in pixels. This operation is applicable only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*

Use the function `ProUICheckbuttonMinimumsizeGet()` to retrieve the minimum size of the width and height of the check button in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUICheckbuttonPositionSet()` to set the position to the checkbutton with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUICheckbuttonPositionGet()` to get the position of the checkbutton with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUICheckbuttonSizeGet()` to get the size of the checkbutton. This operation is applicable only if the parent is a drawing area.

Checkbutton Action Callbacks

Functions Introduced

- **ProUICheckbuttonActivateActionSet()**

The function `ProUICheckbuttonActivateActionSet()` sets the callback action to be invoked when the user toggles the state of the checkbutton.

Drawing Area

Drawing Area Attributes

Attribute Name	Get Function	Set Function(s)
.ArcDirection on page	ProUIDrawingareaArcdirectionGet()	ProUIDrawingareaArcdirectionSet()
.ArcFillMode on page	ProUIDrawingareaArcfillmodeGet()	ProUIDrawingareaArcfillmodeSet()
.AttachBottom on page	ProUIDrawingareaIsAttachedBottom()	ProUIDrawingareaAttachBottom() ProUIDrawingareaUnattachBottom()
.AttachTop on page	ProUIDrawingareaIsAttachedTop()	ProUIDrawingareaAttachTop() ProUIDrawingareaUnattachTop()
.AttachRight on page	ProUIDrawingareaIsAttachedRight()	ProUIDrawingareaAttachRight() ProUIDrawingareaUnattachRight()
.AttachLeft on page	ProUIDrawingareaIsAttachedLeft()	ProUIDrawingareaAttachLeft() ProUIDrawingareaUnattachLeft()
.BackgroundColor on page	ProUIDrawingareaBack	ProUIDrawingareaBack

Attribute Name	Get Function	Set Function(s)
	groundcolorGet()	groundcolorSet()
.BgColor on page	ProUIDrawingareaBgcolorGet()	ProUIDrawingareaBgcolorSet()
.BottomOffset on page	ProUIDrawingareaBottomoffsetGet()	ProUIDrawingareaBottomoffsetSet()
.ChildNames on page	ProUIDrawingareaChildnamesGet()	Not Applicable
.ClipChildren on page	ProUIDrawingareaClipchildrenGet()	ProUIDrawingareaClipchildrenSet()
.Decorated on page	ProUIDrawingareaIsDecorated()	ProUIDrawingareaDecorate() ProUIDrawingareaUndecorate()
.DrawingHeight on page	ProUIDrawingareaDrawingheightGet()	ProUIDrawingareaDrawingheightSet()
.DrawingMode on page	ProUIDrawingareaDrawingmodeGet()	ProUIDrawingareaDrawingmodeSet()
.DrawingWidth on page	ProUIDrawingareaDrawingwidthGet()	ProUIDrawingareaDrawingwidthSet()
.FillMode on page	ProUIDrawingareaFillmodeGet()	ProUIDrawingareaFillmodeSet()
.FontClass on page	ProUIDrawingareaFontclassGet()	ProUIDrawingareaFontclassSet()
.FontSize on page	ProUIDrawingareaFontsizeGet()	ProUIDrawingareaFontsizeSet()
.FontStyle on page	ProUIDrawingareaFontstyleGet()	ProUIDrawingareaFontstyleSet()
.FgColor on page	ProUIDrawingareaFgcolorGet()	ProUIDrawingareaFgcolorSet()
.HelpText on page	ProUIDrawingareaHelptextGet()	ProUIDrawingareaHelptextSet()
.Images on page	ProUIDrawingareaImagesGet()	ProUIDrawingareaImagesSet()
.LeftOffset on page	ProUIDrawingareaLeftoffsetGet()	ProUIDrawingareaLeftoffsetSet()
.LineStyle on page	ProUIDrawingareaLinestyleGet()	ProUIDrawingareaLinestyleSet()
.ParentName on page	ProUIDrawingareaParentnameGet()	Not Applicable
.PolygonFillMode on page	ProUIDrawingareaPolygonfillmodeGet()	ProUIDrawingareaPolygonfillmodeSet()
.PopupMenu on page	ProUIDrawingareaPopupmenuGet()	ProUIDrawingareaPopupmenuSet()
.RightOffset on page	ProUIDrawingareaRightoffsetGet()	ProUIDrawingareaRightoffsetSet()
.Sensitive on page	ProUIDrawingareaIsEnabled()	ProUIDrawingareaEnable() ProUIDrawingareaDisable()

Attribute Name	Get Function	Set Function(s)
.TopOffset on page	ProUIDrawingareaTopoffsetGet()	ProUIDrawingareaTopoffsetSet()
.Tracking on page	ProUIDrawingareaIsTrackingEnabled()	ProUIDrawingareaEnableTracking() ProUIDrawingareaDisableTracking()
.Visible on page	ProUIDrawingareaIsVisible()	ProUIDrawingareaShow() ProUIDrawingareaHide()

Adding and Removing Components

Component Name	Adding Functions	Removing Functions
Checkbutton	ProUIDrawingareaCheckbuttonAdd()	ProUIDrawingareaCheckbuttonRemove()
Drawingarea	ProUIDrawingareaDrawingareaAdd()	ProUIDrawingareaDrawingareaRemove()
Inputpanel	ProUIDrawingareaInputpanelAdd()	ProUIDrawingareaInputpanelRemove()
Label	ProUIDrawingareaLabelAdd()	ProUIDrawingareaLabelRemove()
Layout	ProUIDrawingareaLayoutAdd()	ProUIDrawingareaLayoutRemove()
List	ProUIDrawingareaListAdd()	ProUIDrawingareaListRemove()
Optionmenu	ProUIDrawingareaOptionmenuAdd()	ProUIDrawingareaOptionmenuRemove()
Progressbar	ProUIDrawingareaProgressbarAdd()	ProUIDrawingareaProgressbarRemove()
Pushbutton	ProUIDrawingareaPushbuttonAdd()	ProUIDrawingareaPushbuttonRemove()
Radiogroup	ProUIDrawingareaRadiogroupAdd()	ProUIDrawingareaRadiogroupRemove()
Slider	ProUIDrawingareaSliderAdd()	ProUIDrawingareaSliderRemove()
Spinbox	ProUIDrawingareaSpinboxAdd()	ProUIDrawingareaSpinboxRemove()
Tab	ProUIDrawingareaTabAdd()	ProUIDrawingareaTabRemove()
Table	ProUIDrawingareaTableAdd()	ProUIDrawingareaTableRemove()
Textarea	ProUIDrawingareaTextareaAdd()	ProUIDrawingareaTextareaRemove()
Thumbwheel	ProUIDrawingareaThumbwheelAdd()	ProUIDrawingareaThumbwheelRemove()
Tree	ProUIDrawingareaTreeAdd()	ProUIDrawingareaTreeRemove()

Components added to drawing areas are not managed in grids like components in most other containers. Instead, use the operations that set the component size and position to place the component how you wish in the drawing area.

Note

This means that components may possibly overlap each other in a drawing area depending on their individual placement.

Drawing Area Action Callbacks

Functions Introduced

- **ProUIDrawingareaEnterActionSet()**
- **ProUIDrawingareaExitActionSet()**
- **ProUIDrawingareaMoveActionSet()**
- **ProUIDrawingareaLbuttonarmActionSet()**
- **ProUIDrawingareaLbuttondisarmActionSet()**
- **ProUIDrawingareaLbuttonactivateActionSet()**
- **ProUIDrawingareaLbuttondblclkActionSet()**
- **ProUIDrawingareaMbuttonarmActionSet()**
- **ProUIDrawingareaMbuttondisarmActionSet()**
- **ProUIDrawingareaMbuttondisarmActionSet()**
- **ProUIDrawingareaMbuttonactivateActionSet()**
- **ProUIDrawingareaMbuttondblclkActionSet()**
- **ProUIDrawingareaRbuttonarmActionSet()**
- **ProUIDrawingareaRbuttondisarmActionSet()**
- **ProUIDrawingareaRbuttonactivateActionSet()**
- **ProUIDrawingareaRbuttondblclkActionSet()**
- **ProUIDrawingareaUpdateActionSet()**
- **ProUIDrawingareaResizeActionSet()**
- **ProUIDrawingareaPostmanagenotifyActionSet()**

Use the function `ProUIDrawingareaEnterActionSet()` to set the action function to be called when the user has moved the cursor into the drawing area. This action will be generated only if tracking is enabled for the drawing area.

Use the function `ProUIDrawingareaExitActionSet()` to set the function to be called when the user has moved the cursor out of the drawing area. This action will be generated only if tracking is enabled for the drawing area.

Use the function `ProUIDrawingareaMoveActionSet()` to set the function to be called when the cursor is moved over the drawing area. This action will be generated only if tracking is enabled for the drawing area.

Use the function `ProUIDrawingareaLbuttonarmActionSet()` to set the function to be called when the left mouse button is clicked in the drawing area.

Use the function `ProUIDrawingareaLbuttondisarmActionSet()` to set the function to be called when the left mouse button is released in the drawing area.

Use the function `ProUIDrawingareaLbuttonactivateActionSet()` to set the function to be called when the left mouse button is clicked and released in the drawing area.

Use the function `ProUIDrawingareaLbuttondblclkActionSet()` to set the function to be called when the left mouse button is double-clicked in the drawing area.

Use the function `ProUIDrawingareaMbuttonarmActionSet()` to set the function to be called when the middle mouse button is clicked in the drawing area.

Use the function `ProUIDrawingareaMbuttondisarmActionSet()` to set the function to be called when the middle mouse button is released in the drawing area.

Use the function `ProUIDrawingareaMbuttonactivateActionSet()` to set the function to be called when the middle mouse button is clicked and released in the drawing area.

Use the function `ProUIDrawingareaMbuttondblclkActionSet()` to set the function to be called when the middle mouse button is double-clicked in the drawing area.

Use the function `ProUIDrawingareaRbuttonarmActionSet()` to set the function to be called when the right mouse button is clicked in the drawing area.

Use the function `ProUIDrawingareaRbuttondisarmActionSet()` to set the function to be called when the right mouse button is released in the drawing area.

Use the function `ProUIDrawingareaRbuttonactivateActionSet()` to set the function to be called when the right mouse button is clicked and released in the drawing area.

Use the function `ProUIDrawingareaRbuttondblclkActionSet()` to set the function to be called when the right mouse button is double-clicked in the drawing area.

Use the function `ProUIDrawingareaUpdateActionSet()` to set the function to be called when the drawing area needs to be updated due to a system-level color scheme change.

Use the function `ProUIDrawingareaResizeActionSet()` to set the function to be called when the drawing area is resized.

Note

Any graphics, text or images added to the drawing area is typically cleared after a resize.

Use the function `ProUIDrawingareaPostmanagenotifyActionSet()` to set the function to be called when the drawing area has just been displayed. Use this callback to setup the initial graphics, text, and images in the drawing area.

Drawing Area Operations

Functions Introduced

- **ProUIDrawingareaAnchorSet()**
- **ProUIDrawingareaSizeGet()**
- **ProUIDrawingareaSizeSet()**
- **ProUIDrawingareaMinimumsizeGet()**
- **ProUIDrawingareaPositionGet()**
- **ProUIDrawingareaPositionSet()**
- **ProUIDrawingareaClear()**
- **ProUIDrawingareaCopyArea()**
- **ProUIDrawingareaPointDraw()**
- **ProUIDrawingareaPointsDraw()**
- **ProUIDrawingareaLineDraw()**
- **ProUIDrawingareaLinesDraw()**
- **ProUIDrawingareaPolylineDraw()**
- **ProUIDrawingareaRectDraw()**
- **ProUIDrawingareaRectsDraw()**
- **ProUIDrawingareaRectFill()**
- **ProUIDrawingareaRectsFill()**
- **ProUIDrawingareaShadowRectDraw()**

- **ProUIDrawingareaShadowRectsDraw()**
- **ProUIDrawingareaPolygonDraw()**
- **ProUIDrawingareaPolygonFill()**
- **ProUIDrawingareaArcDraw()**
- **ProUIDrawingareaArcsDraw()**
- **ProUIDrawingareaArcFill()**
- **ProUIDrawingareaArcsFill()**
- **ProUIDrawingareaEllipseDraw()**
- **ProUIDrawingareaEllipsesDraw()**
- **ProUIDrawingareaEllipseFill()**
- **ProUIDrawingareaEllipsesFill()**
- **ProUIDrawingareaImageDraw()**
- **ProUIDrawingareaImagesDraw()**
- **ProUIDrawingareaStringDraw()**
- **ProUIDrawingareaStringsDraw()**
- **ProUIDrawingareaStringsizeGet()**
- **ProUIDrawingareaStringbaselineGet()**
- **ProUIDrawingareaImagesizeGet()**
- **ProUIDrawingareaCursorposGet()**
- **ProUIDrawingareaCursorposSet()**

Use the function `ProUIDrawingareaAnchorSet()` to set the position of the drawing area with respect to a given anchor location. This function is applicable only if the parent of the drawing area is another drawing area.

Use the function `ProUIDrawingareaSizeGet()` to get the size of the drawing area. This operation is applicable only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIDrawingareaMinimumsizeGet()` to retrieve the minimum size of the width and height of the drawing area in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIDrawingareaSizeSet()` to set the size of the drawing area. This operation is applicable only if the parent is a drawing area.

Use the function `ProUIDrawingareaPositionGet()` to get the position of the drawing area with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUIDrawingareaPositionSet()` to set the position to the drawing area with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUIDrawingareaClear()` to clear the contents of the drawing area by painting it in the drawing background color.

Use the function `ProUIDrawingareaCopyArea()` to copy the contents within a given boundary at a location in the drawing area to another location.

Use the function `ProUIDrawingareaPointDraw()` to draw a point in the drawing area.

Use the function `ProUIDrawingareaPointsDraw()` to draw an array of points in the drawing area.

Use the function `ProUIDrawingareaLineDraw()` to draw a line in the drawing area.

Use the function `ProUIDrawingareaLinesDraw()` to draw a set of lines between in the drawing area. Each line will be drawn from the indicated start point in the array to the corresponding endpoint.

Use the function `ProUIDrawingareaPolylineDraw()` to draw a series of connected lines in the drawing area.

Use the function `ProUIDrawingareaRectDraw()` to draw a rectangle in the drawing area.

Use the function `ProUIDrawingareaRectsDraw()` to draw a set of rectangles in the drawing area.

Use the function `ProUIDrawingareaRectFill()` to draw a filled rectangle in the drawing area.

Use the function `ProUIDrawingareaRectsFill()` to draw a set of filled rectangles in the drawing area.

Use the function `ProUIDrawingareaShadowRectDraw()` to draw a rectangle with a shadow border.

Use the function `ProUIDrawingareaShadowRectsDraw()` to draw a set of rectangles with shadow borders.

Use the function `ProUIDrawingareaPolygonDraw()` to draw a polygon in the drawing area.

Use the function `ProUIDrawingareaPolygonFill()` to draw a filled polygon in the drawing area.

Use the function `ProUIDrawingareaArcDraw()` to draw an arc in the drawing area.

Use the function `ProUIDrawingareaArcsDraw()` to draw a set of arcs in the drawing area.

Use the function `ProUIDrawingareaArcFill()` to draw a filled arc in the drawing area.

Use the function `ProUIDrawingareaArcsFill()` to draw a set of filled arcs in the drawing.

Use the function `ProUIDrawingareaEllipseDraw()` to draw an ellipse in the drawing area.

Use the function `ProUIDrawingareaEllipsesDraw()` to draw a set of ellipses in the drawing area.

Use the function `ProUIDrawingareaEllipseFill()` to draw a filled ellipse in the drawing area.

Use the function `ProUIDrawingareaEllipsesFill()` to draw a set of filled ellipses in the drawing area.

Use the function `ProUIDrawingareaImageDraw()` to draw an image in the drawing area.

Use the function `ProUIDrawingareaImagesDraw()` to draw images at the given positions in the drawing area.

Use the function `ProUIDrawingareaStringDraw()` to draw a string at the given position in the drawing area.

Use the function `ProUIDrawingareaStringsDraw()` to draw strings at the given positions in the drawing.

Use the function `ProUIDrawingareaStringsizeGet()` to get the size that the given text string will occupy in the drawing area, given the current drawing area font settings.

Use the function `ProUIDrawingareaStringbaselineGet()` to get the height from the top of the string border to the string baseline for the given text string in the drawing area, given the current drawing area font settings.

Use the function `ProUIDrawingareaImagesizeGet()` to get the size of the image in the drawing area.

Use the function `ProUIDrawingareaCursorposGet()` to get the position of the cursor in the drawing area.

Use the function `ProUIDrawingareaCursorposSet()` to set the cursor at the given location in the drawing area.

Input Panel

Input Panel Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUIInputpanelIsAttachedBottom()	ProUIInputpanelAttachBottom() ProUIInputpanelUnattachBottom()
.AttachTop on page	ProUIInputpanelIsAttachedTop()	ProUIInputpanelAttachTop() ProUIInputpanelUnattachTop()
.AttachRight on page	ProUIInputpanelIsAttachedRight()	ProUIInputpanelAttachRight() ProUIInputpanelUnattachRight()
.AttachLeft on page	ProUIInputpanelIsAttachedLeft()	ProUIInputpanelAttachLeft() ProUIInputpanelUnattachLeft()
.Autohighlight on page	ProUIInputpanelIsAutohighlightEnabled()	ProUIInputpanelAutohighlightEnable() ProUIInputpanelAutohighlightDisable()
.BackgroundColor on page	Not Applicable	ProUIInputpanelBackgroundcolorSet()
.BottomOffset on page	ProUIInputpanelBottomoffsetGet()	ProUIInputpanelBottomoffsetSet()
.Columns on page	ProUIInputpanelColumnsGet()	ProUIInputpanelColumnsSet()
.Denominator on page	ProUIInputpanelDenominatorGet()	ProUIInputpanelDenominatorSet()
.Digits on page	ProUIInputpanelDigitsGet()	ProUIInputpanelDigitsSet()
.Double on page	ProUIInputpanelDoubleGet()	ProUIInputpanelDoubleSet()
.DoubleFormat on page	ProUIInputpanelDoubleformatGet()	ProUIInputpanelDoubleformatSet()
.Editable on page	ProUIInputpanelIsEditable()	ProUIInputpanelEditable() ProUIInputpanelReadOnly()
.HelpText on page	ProUIInputpanelHelptextGet()	ProUIInputpanelHelptextSet()
.InputType on page	ProUIInputpanelInputtype	ProUIInputpanelInputtype

Attribute Name	Get Function	Set Function(s)
	Get()	Set()
.Integer on page	ProUIInputpanelIntegerGet()	ProUIInputpanelIntegerSet()
.LeftOffset on page	ProUIInputpanelLeftoffsetGet()	ProUIInputpanelLeftoffsetSet()
.MinColumns on page	ProUIInputpanelMincolumnsGet()	ProUIInputpanelMincolumnsSet()
.MinDouble on page	ProUIInputpanelMindoubleGet()	ProUIInputpanelMindoubleSet()
.MaxDouble on page	ProUIInputpanelMaxdoubleGet()	ProUIInputpanelMaxdoubleSet()
.MinInteger on page	ProUIInputpanelMinintegerGet()	ProUIInputpanelMinintegerSet()
.MaxInteger on page	ProUIInputpanelMaxintegerGet()	ProUIInputpanelMaxintegerSet()
.MaxLen on page	ProUIInputpanelMaxlenGet()	ProUIInputpanelMaxlenSet()
.Numerator on page	ProUIInputpanelNumeratorGet()	ProUIInputpanelNumeratorSet()
.Ordinal on page	ProUIInputpanelOrdinalGet()	ProUIInputpanelOrdinalSet()
.ParentName on page	ProUIInputpanelParentnameGet()	Not Applicable
.Password on page	ProUIInputpanelPasswordcharGet()	ProUIInputpanelUsePasswordchars() ProUIInputpanelUseNormalchars() ProUIInputpanelContainsPassword() ProUIInputpanelPasswordcharSet()
.PopupMenu on page	ProUIInputpanelPopupmenuGet()	ProUIInputpanelPopupmenuSet()
.RightOffset on page	ProUIInputpanelRightoffsetGet()	ProUIInputpanelRightoffsetSet()
.Sensitive on page	ProUIInputpanelIsEnabled()	ProUIInputpanelEnable() ProUIInputpanelDisable()
.String on page	ProUIInputpanelStringGet()	ProUIInputpanelStringSet()
.TabCharsAllow on page	ProUIInputpanelTabcharsAllow()	ProUIInputpanelTabcharsDisallow() ProUIInputpanelAllowsTabchars()
.TopOffset on page	ProUIInputpanelTopoffsetGet()	ProUIInputpanelTopoffsetSet()
.Value on page	ProUIInputpanelValueGet()	ProUIInputpanelValueSet()

Attribute Name	Get Function	Set Function(s)
.Visible on page	ProUIInputpanelIsVisible()	ProUIInputpanelShow() ProUIInputpanelHide()
.WideString on page	ProUIInputpanelWidestringGet()	ProUIInputpanelWidestringSet()

Input Panel Action Callbacks

Functions Introduced

- **ProUIInputpanelActivateActionSet()**
- **ProUIInputpanelFocusinActionSet()**
- **ProUIInputpanelFocusoutActionSet()**
- **ProUIInputpanelInputActionSet()**

Use the function `ProUIInputpanelActivateActionSet()` to set the action callback to be called when the user hits return in an input panel.

Use the function `ProUIInputpanelFocusinActionSet()` to set the focus in action for an input panel. This function is called when the user moves the cursor onto the input panel using the mouse or `<TAB>` key.

Use the function `ProUIInputpanelFocusoutActionSet()` to set the focus out action for an input panel. This function is called when the user moves the cursor off of the input panel using the mouse or `<TAB>` key.

Use the function `ProUIInputpanelInputActionSet()` to set the action callback to be called when the user enters a key in an input panel.

Input Panel Operations

Functions Introduced

- **ProUIInputpanelAnchorSet()**
- **ProUIInputpanelSizeGet()**
- **ProUIInputpanelSizeSet()**
- **ProUIInputpanelMinimumsizeGet()**
- **ProUIInputpanelPositionGet()**
- **ProUIInputpanelPositionSet()**

Use the function `ProUIInputpanelAnchorSet()` to set the position of the input panel with respect to a given anchor location. This function is applicable only if the parent of the input panel is a drawing area.

Use the function `ProUIInputpanelSizeGet()` to get the size of the input panel. This operation is applicable only if the parent is a drawing area.

Use the function `ProUIInputpanelSizeSet()` to set the size of the input panel. This operation is applicable only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIInputpanelMinimumsizeGet()` to retrieve the minimum size of the width and height of the input panel in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIInputpanelPositionGet()` to get the position of the input panel with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUIInputpanelPositionSet()` to set the position to the input panel with respect to its parent. This operation is applicable only if the parent is a drawing area.

Label

Label Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUILabelIsAttachedBottom()	ProUILabelAttachBottom() ProUILabelUnattachBottom()
.AttachTop on page	ProUILabelIsAttachedTop()	ProUILabelAttachTop() ProUILabelUnattachTop()
.AttachRight on page	ProUILabelIsAttachedRight()	ProUILabelAttachRight() ProUILabelUnattachRight()
.AttachLeft on page	ProUILabelIsAttachedLeft()	ProUILabelAttachLeft() ProUILabelUnattachLeft()
.Bitmap on page	ProUILabelBitmapGet()	ProUILabelBitmapSet()
.BottomOffset on page	ProUILabelBottomoffsetGet()	ProUILabelBottomoffsetSet()
.Columns on page	ProUILabelColumnsGet()	ProUILabelColumnsSet()
.HelpText on page	ProUILabelHelptextGet()	ProUILabelHelptextSet()
.Label on page	ProUILabelTextGet()	ProUILabelTextSet()
.LeftOffset on page	ProUILabelLeftoffsetGet()	ProUILabelLeftoffsetSet()
.ParentName on page	ProUILabelParentnameGet()	Not Applicable
.PopupMenu on page	ProUILabelPopupmenuGet()	ProUILabelPopupmenuSet()

Attribute Name	Get Function	Set Function(s)
.Resizeable on page	ProUILabelIsResizeable()	ProUILabelEnableResizing() ProUILabelDisableResizing()
.RightOffset on page	ProUILabelRightoffsetGet()	ProUILabelRightoffsetSet()
.Sensitive on page	ProUILabelIsEnabled()	ProUILabelEnable() ProUILabelDisable()
.TopOffset on page	ProUILabelTopoffsetGet()	ProUILabelTopoffsetSet()
.Visible on page	ProUILabelIsVisible()	ProUILabelShow() ProUILabelHide()

Label Operations

Functions Introduced

- **ProUILabelAnchorSet()**
- **ProUILabelSizeSet()**
- **ProUILabelMinimumsizeGet()**
- **ProUILabelPositionGet()**
- **ProUILabelPositionSet()**
- **ProUILabelSizeGet()**

Use the function `ProUILabelAnchorSet()` to set the location to position the label with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUILabelSizeSet()` to set the size of the label. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUILabelMinimumsizeGet()` to retrieve the minimum size of the width and height of the label in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUILabelPositionGet()` to get the position of the label with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUILabelPositionSet()` to set the position to the label with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUILabelSizeGet()` to get the size of the label. This field is used only if the parent is a drawing area.

Layout

Layout Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUILayoutIsAttachedBottom()	ProUILayoutAttachBottom() ProUILayoutUnattachBottom()
.AttachTop on page	ProUILayoutIsAttachedTop()	ProUILayoutAttachTop() ProUILayoutUnattachTop()
.AttachRight on page	ProUILayoutIsAttachedRight()	ProUILayoutAttachRight() ProUILayoutUnattachRight()
.AttachLeft on page	ProUILayoutIsAttachedLeft()	ProUILayoutAttachLeft() ProUILayoutUnattachLeft()
.Bitmap on page	ProUILayoutBitmapGet()	ProUILayoutBitmapSet()
.BottomOffset on page	ProUILayoutBottomoffsetGet()	ProUILayoutBottomoffsetSet()
.ChildNames on page	ProUILayoutChildnamesGet()	Not Applicable
.HelpText on page	ProUILayoutHelptextGet()	ProUILayoutHelptextSet()
.Label on page	ProUILayoutTextGet()	ProUILayoutTextSet()
.LeftOffset on page	ProUILayoutLeftoffsetGet()	ProUILayoutLeftoffsetSet()
.Mapped on page	ProUILayoutIsMapped()	ProUILayoutMappedSet() ProUILayoutMappedUnset()
.ParentName on page	ProUILayoutParentnameGet()	Not Applicable
.PopupMenu on page	ProUILayoutPopupmenuGet()	ProUILayoutPopupmenuSet()
.RightOffset on page	ProUILayoutRightoffsetGet()	ProUILayoutRightoffsetSet()
.Sensitive on page	ProUILayoutIsEnabled()	ProUILayoutEnable() ProUILayoutDisable()
.TopOffset on page	ProUILayoutTopoffsetGet()	ProUILayoutTopoffsetSet()
.Visible on page	ProUILayoutIsVisible()	ProUILayoutShow() ProUILayoutHide()

Adding and Removing Components

Component Name	Adding Functions	Removing Functions
Checkbutton	ProUILayoutCheckbuttonAdd()	ProUILayoutCheckbuttonRemove()
Drawingarea	ProUILayoutDrawingareaAdd()	ProUILayoutDrawingareaRemove()
Inputpanel	ProUILayoutInputpane	ProUILayoutInputpanelRe

Component Name	Adding Functions	Removing Functions
	lAdd()	move()
Label	ProUILayoutLabelAdd()	ProUILayoutLabelRemove()
Layout	ProUILayoutLayoutAdd()	ProUILayoutLayoutRemove()
List	ProUILayoutListAdd()	ProUILayoutListRemove()
Optionmenu	ProUILayoutOptionmenuAdd()	ProUILayoutOptionmenuRemove()
Progressbar	ProUILayoutProgressbarAdd()	ProUILayoutProgressbarRemove()
Pushbutton	ProUILayoutPushbuttonAdd()	ProUILayoutPushbuttonRemove()
Radiogroup	ProUILayoutRadiogroupAdd()	ProUILayoutRadiogroupRemove()
Separator	ProUILayoutSeparatorAdd()	ProUILayoutSeparatorRemove()
Slider	ProUILayoutSliderAdd()	ProUILayoutSliderRemove()
Spinbox	ProUILayoutSpinboxAdd()	ProUILayoutSpinboxRemove()
Tab	ProUILayoutTabAdd()	ProUILayoutTabRemove()
Table	ProUILayoutTableAdd()	ProUILayoutTableRemove()
Textarea	ProUILayoutTextareaAdd()	ProUILayoutTextareaRemove()
Thumbwheel	ProUILayoutThumbwheelAdd()	ProUILayoutThumbwheelRemove()
Tree	ProUILayoutTreeAdd()	ProUILayoutTreeRemove()

See the section on Adding and Removing Components on page 402 for description of how to use the `ProUIGridopts` arguments when adding components to a layout.

Layout Operations

Functions Introduced

- **ProUILayoutAnchorSet()**
- **ProUILayoutSizeSet()**
- **ProUILayoutMinimumsizeGet()**
- **ProUILayoutPositionGet()**
- **ProUILayoutPositionSet()**
- **ProUILayoutSizeGet()**
- **ProUILayoutIsMapped()**

- **ProUILayoutMappedSet()**
- **ProUILayoutMappedUnset()**

Use the function `ProUILayoutAnchorSet()` to set the position of the layout with respect to a given anchor location. This function is applicable only if the parent of the layout is a drawing area.

Use the function `ProUILayoutSizeSet()` to set the size of the layout. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUILayoutMinimumsizeGet()` to retrieve the minimum size of the width and height of the layout in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUILayoutPositionGet()` to get the position of the layout with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUILayoutPositionSet()` to set the position to the layout with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUILayoutSizeGet()` to get the size of the layout. This field is used only if the parent is a drawing area.

The function `ProUILayoutIsMapped()` specifies if the given layout component is mapped. The value of the output flag is `PRO_B_TRUE` if the layout is mapped, else it is `PRO_B_FALSE`.

The functions `ProUILayoutShow()` and `ProUILayoutHide()` show and hide the layout component respectively, along with its contents. Use the function `ProUILayoutMappedSet()` to keep the size of the layout unchanged while working with these functions. To collapse or expand the layout to its nominal size, use the function `ProUILayoutMappedUnset()`.

List

List Attributes

Attribute Name	Get Function	Set Function(s)
`.AttachBottom` on page	`ProUIListIsAttachedBottom()`	`ProUIListAttachBottom()` `ProUIListUnattachBottom()`
`.AttachTop` on page	`ProUIListIsAttachedTop()`	`ProUIListAttachTop()` `ProUIListUnattachTop()`
`.AttachRight` on page	`ProUIListIsAttachedRight()`	`ProUIListAttachRight()` `ProUIListUnattachRight()`

Attribute Name	Get Function	Set Function(s)
.AttachLeft on page	ProUIListIsAttached Left()	ProUIListAttachLeft() ProUIListUnattachLeft()
.BackgroundColor on page	Not Applicable	ProUIListBackgroundcolorSet()
.BottomOffset on page	ProUIListBottomoffset Get()	ProUIListBottomoffsetSet()
.Columns on page	ProUIListColumnsGet()	ProUIListColumnsSet()
.ColumnLabel on page	ProUIListColumnlabel Get()	ProUIListColumnlabelSet()
.HelpText on page	ProUIListHelptextGet()	ProUIListHelptextSet()
.ItemHelpText on page	ProUIListItemhelptext Get()	ProUIListItemhelptextSet()
.ItemImage on page	ProUIListItemimageGet()	ProUIListItemimageSet()
.Label on page	ProUIListLabelsGet()	ProUIListLabelsSet()
.Lastentereditem on page	ProUIListLastenteredi temGet()	Not Applicable
.LeftOffset on page	ProUIListLeftoffset Get()	ProUIListLeftoffsetSet()
.ListState on page	ProUIListStateGet()	ProUIListStateSet()
.ListType on page	ProUIListListtypeGet()	ProUIListListtypeSet()
.ParentName on page	ProUIListParentname Get()	Not Applicable
.MinColumns on page	ProUIListMincolumns Set()	ProUIListMincolumnsGet()
.MinRows on page	ProUIListMinrowsGet()	ProUIListMinrowsSet()
.Names on page	ProUIListNamesGet()	ProUIListNamesSet()
.PopupMenu on page	ProUIListPopupmenuGet()	ProUIListPopupmenuSet()
.RightOffset on page	ProUIListRightoffset Get()	ProUIListRightoffsetSet()
.SelectedNames on page	ProUIListSelectednames Get()	ProUIListSelectednamesSet()
.SelectionPolicy on page	ProUIListSelectionpoli cyGet()	ProUIListSelectionpolicySet()
.Sensitive on page	ProUIListIsEnabled()	ProUIListEnable() ProUIListDisable()
.TopOffset on page	ProUIListTopoffsetGet()	ProUIListTopoffsetSet()
.Visible on page	ProUIListIsVisible()	ProUIListShow() ProUIListHide()
.VisibleRows on page	ProUIListVisiblerows Get()	ProUIListVisiblerowsSet()

List Action Callbacks

Functions Introduced

- **ProUIListActivateActionSet()**
- **ProUIListSelectActionSet()**
- **ProUIListTriggerhighlightActionSet()**
- **ProUIListFocusinActionSet()**
- **ProUIListFocusoutActionSet()**

Use the function `ProUIListActivateActionSet()` to set the activate action for a list. This function is called when the return key is pressed or the mouse is double-clicked in the list.

Use the function `ProUIListSelectActionSet()` to set the select action for a list component.

Use the function `ProUIListTriggerhighlightActionSet()` to set the trigger highlight action for a list. This function is called when the user moves the mouse over an item on the list.

Use the function `ProUIListFocusinActionSet()` to set the focus in action for a list. This function is called when the user moves the cursor onto of the list using the mouse or `<TAB>` key.

Use the function `ProUIListFocusoutActionSet()` to set the focus out action for a list. This function is called when the user moves the cursor off of the list using the mouse or `<TAB>` key.

List Operations

Functions Introduced

- **ProUIListAnchorSet()**
- **ProUIListSizeSet()**
- **ProUIListMinimumsizeGet()**
- **ProUIListPositionGet()**
- **ProUIListPositionSet()**
- **ProUIListSizeGet()**
- **ProUIListStateGet()**
- **ProUIListStateSet()**

Use the function `ProUIListAnchorSet()` to set the position of the list with respect to a given anchor location. This function is applicable only if the parent of the list is a drawing area.

Use the function `ProUIListSizeSet()` to set the size of the list. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIListMinimumsizeGet()` to retrieve the minimum size of the width and height of the list in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIListPositionSet()` to set the position to the list with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIListPositionGet()` to get the position of the list with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIListSizeGet()` to get the size of the list. This field is used only if the parent is a drawing area.

Use the function `ProUIlistStateGet()` to get the state of the item in the list. The state is applicable only for a "check" type of list and refers to the checked or unchecked status of the item.

Use the function `ProUIListStateSet()` to set the state of the item in the list. The state is applicable only for a "check" type of list and refers to the checked or unchecked status of the item.

Example 2: To use UI List Functions

The sample code in `UgUIListImplement.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui` shows the source code for UI List functions. The application gets the names of the parts that constitute any given drawing and then populates the list area in a newly created dialog with those names.

Menubar

Menubar Attributes

Attribute Name	Get Function	Set Function(s)
`.AttachBottom` on page	`ProUIMenubarIsAttachedBottom()`	`ProUIMenubarAttachBottom()` `ProUIMenubarUnattachBottom()`
`.AttachTop` on page	`ProUIMenubarIsAttachedTop()`	`ProUIMenubarAttachTop()` `ProUIMenubarUnattachTop()`
`.AttachRight` on page	`ProUIMenubarIsAttachedRight()`	`ProUIMenubarAttachRight()` `ProUIMenubarUnattachRight()`
`.AttachLeft` on page	`ProUIMenubarIsAttachedLeft()`	`ProUIMenubarAttachLeft()` `ProUIMenubarUnattachLeft()`
`.BottomOffset` on page	`ProUIMenubarBottomoffsetGet()`	`ProUIMenubarBottomoffsetSet()`
`.HelpText` on page	`ProUIMenubarHelptextGet()`	`ProUIMenubarHelptextSet()`

Attribute Name	Get Function	Set Function(s)
.ItemHelpText on page	ProUIMenubarItemhelptextGet()	ProUIMenubarItemhelptextSet()
.Label on page	ProUIMenupaneTextGet()	ProUIMenupaneTextSet()
.LeftOffset on page	ProUIMenubarLeftoffsetGet()	ProUIMenubarLeftoffsetSet()
.Names on page	ProUIMenubarNamesGet()	ProUIMenubarNamesSet()
.PopupMenu on page	ProUIMenubarPopupmenuGet()	ProUIMenubarPopupmenuSet()
.RightOffset on page	ProUIMenubarRightoffsetGet()	ProUIMenubarRightoffsetSet()
.SelectableNames on page	ProUIMenubarSelectablenamesGet()	ProUIMenubarSelectablenamesSet()
.Sensitive on page	ProUIMenubarIsEnabled()	ProUIMenubarEnable() ProUIMenubarDisable()
.TopOffset on page	ProUIMenubarTopoffsetGet()	ProUIMenubarTopoffsetSet()
.Visible on page	ProUIMenubarIsVisible()	ProUIMenubarShow() ProUIMenubarHide()
.VisibleNames on page	ProUIMenubarVisiblenamesGet()	ProUIMenubarVisiblenamesSet()

Menupane

Menupane Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUIMenupaneIsAttachedBottom()	ProUIMenupaneAttachBottom() ProUIMenupaneUnattachBottom()
.AttachTop on page	ProUIMenupaneIsAttachedTop()	ProUIMenupaneAttachTop() ProUIMenupaneUnattachTop()
.AttachRight on page	ProUIMenupaneIsAttachedRight()	ProUIMenupaneAttachRight() ProUIMenupaneUnattachRight()
.AttachLeft on page	ProUIMenupaneIsAttachedLeft()	ProUIMenupaneAttachLeft() ProUIMenupaneUnattachLeft()
.Bitmap on page	ProUIMenupaneBitmapGet()	ProUIMenupaneBitmapSet()
.BottomOffset on page	ProUIMenupaneBottomoffsetGet()	ProUIMenupaneBottomoffsetSet()

Attribute Name	Get Function	Set Function(s)
.ChildNames on page	ProUIMenupaneChildnamesGet()	Not Applicable
.LeftOffset on page	ProUIMenupaneLeftoffsetGet()	ProUIMenupaneLeftoffsetSet()
.PopupMenu on page	ProUIMenupanePopupmenuGet()	ProUIMenupanePopupmenuSet()
.RightOffset on page	ProUIMenupaneRightoffsetGet()	ProUIMenupaneRightoffsetSet()
.TopOffset on page	ProUIMenupaneTopoffsetGet()	ProUIMenupaneTopoffsetSet()
.Visible on page	ProUIMenupaneIsVisible()	ProUIMenupaneShow() ProUIMenupaneHide()

Adding and Removing Components

Component Name	Adding Functions	Removing Functions
Cascadebutton	ProUIMenupaneCascadebuttonAdd() ProUIMenupaneCascadebuttonInsert()	ProUIMenupaneCascadebutonRemove()
Checkbutton	ProUIMenupaneCheckbuttonAdd() ProUIMenupaneCheckbuttonInsert()	ProUIMenupaneCheckbuttonRemove()
Pushbutton	ProUIMenupanePushbuttonAdd() ProUIMenupanePushbuttonInsert()	ProUIMenupanePushbuttonRemove()
Radiogroup	ProUIMenupaneRadiogroupAdd() ProUIMenupaneRadiogroupInsert()	ProUIMenupaneRadiogroupRemove()
Separator	ProUIMenupaneSeparatorAdd() ProUIMenupaneSeparatorInsert()	ProUIMenupaneSeparatorRemove()

Optionmenu

Optionmenu Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUIOptionmenuIsAttachedBottom()	ProUIOptionmenuAttachBottom() ProUIOptionmenuUnattachBottom()
.AttachTop on page	ProUIOptionmenuIsAttachedTop()	ProUIOptionmenuAttachTop() ProUIOptionmenuUnattachTop()
.AttachRight on page	ProUIOptionmenuIsAttachedRight()	ProUIOptionmenuAttachRight() ProUIOptionmenuUnattachRight()
.AttachLeft on page	ProUIOptionmenuIsAttachedLeft()	ProUIOptionmenuAttachLeft() ProUIOptionmenuUnattachLeft()
.BottomOffset on page	ProUIOptionmenuBottomoffsetGet()	ProUIOptionmenuBottomoffsetSet()
.Columns on page	ProUIOptionmenuColumnsGet()	ProUIOptionmenuColumnsSet()
.Editable on page	ProUIOptionmenuIsEditable()	ProUIOptionmenuEditable() ProUIOptionmenuReadOnly()
.HelpText on page	ProUIOptionmenuHelptextGet()	ProUIOptionmenuHelptextSet()
.ItemHelpText on page	ProUIOptionmenuItemhelptextGet()	ProUIOptionmenuItemhelptextSet()
.ItemImage on page	ProUIOptionmenuItemimageGet()	ProUIOptionmenuItemimageSet()
.Label on page	ProUIOptionmenuLabelsGet()	ProUIOptionmenuLabelsSet()
.Lastenterededitem on page	ProUIOptionmenuLastentereditemGet()	Not Applicable
.LeftOffset on page	ProUIOptionmenuLeftoffsetGet()	ProUIOptionmenuLeftoffsetSet()
.MinColumns on page	ProUIOptionmenuMincolumnsGet()	ProUIOptionmenuMincolumnsSet()
.Names on page	ProUIOptionmenuNamesGet()	ProUIOptionmenuNamesSet()
.ParentName on page	ProUIOptionmenuParentnameGet()	Not Applicable
.PopupMenu on page	ProUIOptionmenuPopupmenuGet()	ProUIOptionmenuPopupmenuSet()
.RightOffset on page	ProUIOptionmenuRightoffsetGet()	ProUIOptionmenuRightoffsetSet()
.SelectedNames on page	ProUIOptionmenuSelectednamesGet()	ProUIOptionmenuSelectednamesSet()

Attribute Name	Get Function	Set Function(s)
.Sensitive on page	ProUIOptionmenuIsEnabled()	ProUIOptionmenuEnable() ProUIOptionmenuDisable()
.TopOffset on page	ProUIOptionmenuTopoffset Get()	ProUIOptionmenuTopoffset Set()
.Value on page	ProUIOptionmenuValueGet()	ProUIOptionmenuValueSet()
.Visible on page	ProUIOptionmenuIsVisible()	ProUIOptionmenuShow() ProUIOptionmenuHide()
.VisibleRows on page	ProUIOptionmenuVisiblerows Get()	ProUIOptionmenuVisiblerows Set()

Optionmenu Action Callbacks

Functions Introduced

- **ProUIOptionmenuActivateActionSet()**
- **ProUIOptionmenuSelectActionSet()**
- **ProUIOptionmenuInputActionSet()**
- **ProUIOptionmenuTriggerhighlightActionSet()**
- **ProUIOptionmenuFocusinActionSet()**
- **ProUIOptionmenuFocusoutActionSet()**

Use the function `ProUIOptionmenuActivateActionSet()` to set the activate action for an option menu. This function is called when the user modifies the contents of the option menu. The option menu must be editable. The action callback is called when you press the ENTER key in the input panel of the option menu.

Use the function `ProUIOptionmenuSelectActionSet()` to set the select action for a option menu component.

Use the function `ProUIOptionmenuInputActionSet()` to set the input action for an optionmenu. This function is called when the user changes the contents of the option menu. This is only valid for editable optionmenus.

Use the function `ProUIOptionmenuTriggerhighlightActionSet()` to set the trigger highlight action for an optionmenu. This function is called when the user moves the mouse on an item in the drop down list of the optionmenu.

Use the function `ProUIOptionmenuFocusinActionSet()` to set the focus in action for an optionmenu.

Use the function `ProUIOptionmenuFocusoutActionSet()` to set the focus out action for an optionmenu.

Optionmenu Operations

Functions Introduced

- **ProUIOptionmenuAnchorSet()**
- **ProUIOptionmenuSizeSet()**
- **ProUIOptionmenuMinimumsizeGet()**
- **ProUIOptionmenuPositionSet()**
- **ProUIOptionmenuPositionGet()**
- **ProUIOptionmenuSizeGet()**

Use the function `ProUIOptionmenuAnchorSet()` to the position of the option menu with respect to a given anchor location. This function is applicable only if the parent of the option menu is a drawing area.

Use the function `ProUIOptionmenuSizeSet()` to set the size of the optionmenu. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIOptionmenuMinimumsizeGet()` to retrieve the minimum size of the width and height of the option menu in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIOptionmenuPositionSet()` to set the position to the optionmenu with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIOptionmenuPositionGet()` to get the position of the optionmenu with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIOptionmenuSizeGet()` to get the size of the optionmenu. This field is used only if the parent is a drawing area.

Progressbar

Progressbar Attributes

Attribute Name	Get Function	Set Function(s)

`.AttachBottom` on page	`ProUIProgressbarIsAttachedBottom()`	`ProUIProgressbarAttachBottom()`
		
		`ProUIProgressbarUnattachBottom()`

`.AttachTop` on page	`ProUIProgressbarIsAttachedTop()`	`ProUIProgressbarAttachTop()`

Attribute Name	Get Function	Set Function(s)
		ProUIProgressbarUnattachTop()
.AttachRight on page	ProUIProgressbarIsAttachedRight()	ProUIProgressbarAttachRight() ProUIProgressbarUnattachRight()
.AttachLeft on page	ProUIProgressbarIsAttachedLeft()	ProUIProgressbarAttachLeft() ProUIProgressbarUnattachLeft()
.BottomOffset on page	ProUIProgressbarBottomoffsetGet()	ProUIProgressbarBottomoffsetSet()
.HelpText on page	ProUIProgressbarHelptextGet()	ProUIProgressbarHelptextSet()
.Integer on page	ProUIProgressbarIntegerGet()	ProUIProgressbarIntegerSet()
.LeftOffset on page	ProUIProgressbarLeftoffsetGet()	ProUIProgressbarLeftoffsetSet()
.Length on page	ProUIProgressbarLengthGet()	ProUIProgressbarLengthSet()
.MaxInteger on page	ProUIProgressbarMaxintegerGet()	ProUIProgressbarMaxintegerSet()
.MinInteger on page	ProUIProgressbarMinintegerGet()	ProUIProgressbarMinintegerSet()
Orientation on page	ProUIProgressbarOrientationGet()	ProUIProgressbarOrientationSet()
.ParentName on page	ProUIProgressbarParentnameGet()	Not Applicable
.PopupMenu on page	ProUIProgressbarPopupmenuGet()	ProUIProgressbarPopupmenuSet()
.ProgressStyle on page	ProUIProgressbarProgressstyleGet()	ProUIProgressbarProgressstyleSet()
.RightOffset on page	ProUIProgressbarRightoffsetGet()	ProUIProgressbarRightoffsetSet()
.TopOffset on page	ProUIProgressbarTopoffsetGet()	ProUIProgressbarTopoffsetSet()
.Visible on page	ProUIProgressbarIsVisible()	ProUIProgressbarShow() ProUIProgressbarHide()

Progressbar Operations

Functions Introduced

- **ProUIProgressbarAnchorSet()**
- **ProUIProgressbarSizeSet()**
- **ProUIProgressbarMinimumsizeGet()**

- **ProUIProgressbarPositionSet()**
- **ProUIProgressbarPositionGet()**
- **ProUIProgressbarSizeGet()**

Use the function `ProUIProgressbarAnchorSet()` to set the position of the progressbar with respect to a given anchor location. This function is applicable only if the parent of the progressbar is a drawing area.

Use the function `ProUIProgressbarSizeSet()` to set the size of the progressbar. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIProgressbarMinimumsizeGet()` to retrieve the minimum size of the width and height of the progress bar in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIProgressbarPositionSet()` to set the position to the progressbar with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIProgressbarPositionGet()` to get the position to the progressbar with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIProgressbarSizeGet()` to get the size of the progressbar. This field is used only if the parent is a drawing area.

Pushbutton

Pushbutton Attributes

Attribute Name	Get Function	Set Function(s)
`.AttachBottom` on page	`ProUIPushbuttonIsAttachedBottom()`	`ProUIPushbuttonAttachBottom()` `ProUIPushbuttonUnattachBottom`
`.AttachTop` on page	`ProUIPushbuttonIsAttachedTop()`	`ProUIPushbuttonAttachTop()` `ProUIPushbuttonUnattachTop()`
`.AttachRight` on page	`ProUIPushbuttonIsAttachedRight()`	`ProUIPushbuttonAttachRight()` `ProUIPushbuttonUnattachRight()`
`.AttachLeft` on page	`ProUIPushbuttonIsAttachedLeft()`	`ProUIPushbuttonAttachLeft()`

Attribute Name	Get Function	Set Function(s)
		ProUIPushbuttonUnattachLeft()
.BottomOffset on page	ProUIPushbuttonBottomoffsetGet()	ProUIPushbuttonBottomoffsetSet()
.Bitmap on page	ProUIPushbuttonBitmapGet()	ProUIPushbuttonBitmapSet()
.ButtonStyle on page	ProUIPushbuttonButtonstyleGet()	ProUIPushbuttonButtonstyleSet()
.HelpText on page	ProUIPushbuttonHelptextGet()	ProUIPushbuttonHelptextSet()
.Label on page	ProUIPushbuttonTextGet()	ProUIPushbuttonTextSet()
.ModalOverride on page	Not Applicable	ProUIPushbuttonModaloverrideSet()
.LeftOffset on page	ProUIPushbuttonLeftoffsetGet()	ProUIPushbuttonLeftoffsetSet()
.ParentName on page	ProUIPushbuttonParentnameGet()	Not Applicable
.PopupMenu on page	ProUIPushbuttonPopupmenuGet()	ProUIPushbuttonPopupmenuSet()
.Resizeable on page	ProUIPushbuttonIsResizeable()	ProUIPushbuttonEnableResizing() ProUIPushbuttonDisableResizing()
.RightOffset on page	ProUIPushbuttonRightoffsetGet()	ProUIPushbuttonRightoffsetSet()
.Sensitive on page	ProUIPushbuttonIsEnabled()	ProUIPushbuttonEnable() ProUIPushbuttonDisable()
.TopOffset on page	ProUIPushbuttonTopoffsetGet()	ProUIPushbuttonTopoffsetSet()
.Visible on page	ProUIPushbuttonIsVisible()	ProUIPushbuttonShow() ProUIPushbuttonHide()

Pushbutton Operations

Functions Introduced

- **ProUIPushbuttonAnchorSet()**
- **ProUIPushbuttonSizeSet()**
- **ProUIPushbuttonPositionSet()**
- **ProUIPushbuttonPositionGet()**
- **ProUIPushbuttonSizeGet()**

Use the function `ProUIPushbuttonAnchorSet()` to set the position of the pushbutton with respect to a given anchor location. This function is applicable only if the parent of the Pushbutton is a drawing area.

Use the function `ProUIPushbuttonSizeSet()` to set the size of the pushbutton. This field is used only if the parent is a drawing area.

Use the function `ProUIPushbuttonPositionSet()` to set the position to the pushbutton with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIPushbuttonPositionGet()` to get the position to the pushbutton with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIPushbuttonSizeGet()` to get the size of the pushbutton. This field is used only if the parent is a drawing area.

Pushbutton Action Callbacks

Function Introduced

- **ProUIPushbuttonActivateActionSet()**

Use the function `ProUIPushbuttonActivateActionSet()` to set the activate action for a pushbutton. This function is called when the user selects the pushbutton.

Example 3: Controlling Component Visibility or Sensitivity at Runtime

The sample code in `UgUIVisibility.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui` shows how to control component visibility or sensitivity at runtime.

Radiogroup

Radiogroup Attributes

Attribute Name	Get Function	Set Function(s)
`.AttachBottom` on page	`ProUIRadiogroupIsAttachedBottom()`	`ProUIRadiogroupAttachBottom()` `ProUIRadiogroupUnattachBottom`
`.AttachTop` on page	`ProUIRadiogroupIsAttachedTop()`	`ProUIRadiogroupAttachTop()` `ProUIRadiogroupUnattachTop()`
`.AttachRight` on page	`ProUIRadiogroupIsAttachedRight()`	`ProUIRadiogroupAttachRight()` `ProUIRadiogroupUnattachRight()`
`.AttachLeft` on page	`ProUIRadiogroupIsAttachedLeft()`	`ProUIRadiogroupAttachLeft()` `ProUIRadiogroupUnattachLeft()`

Attribute Name	Get Function	Set Function(s)
.BottomOffset on page	ProUIRadiogroupBottomoffsetGet()	ProUIRadiogroupBottomoffsetSet()
.ButtonStyle on page	ProUIRadiogroupButtonstyleGet()	ProUIRadiogroupButtonstyleSet()
.HelpText on page	ProUIRadiogroupHelptextGet()	ProUIRadiogroupHelptextSet()
.ItemHelpText on page	ProUIRadiogroupItemhelptextGet()	ProUIRadiogroupItemhelptextSet()
.ItemImage on page	ProUIRadiogroupItemimageGet()	ProUIRadiogroupItemimageSet()
.Label on page	ProUIRadiogroupLabelsGet()	ProUIRadiogroupLabelsSet()
.LeftOffset on page	ProUIRadiogroupLeftoffsetGet()	ProUIRadiogroupLeftoffsetSet()
.Names on page	ProUIRadiogroupNamesGet()	ProUIRadiogroupNamesSet()
Orientation on page	ProUIRadiogroupOrientationGet()	ProUIRadiogroupOrientationSet()
.ParentName on page	ProUIRadiogroupParentnameGet()	Not Applicable
.PopupMenu on page	ProUIRadiogroupPopupmenuGet()	ProUIRadiogroupPopupmenuSet()
.Resizeable on page	ProUIRadiogroupIsResizeable()	ProUIRadiogroupEnableResizing() ProUIRadiogroupDisableResizing()
.RightOffset on page	ProUIRadiogroupRightoffsetGet()	ProUIRadiogroupRightoffsetSet()
.SelectedNames on page	ProUIRadiogroupSelectednamesGet()	ProUIRadiogroupSelectednamesSet()
.Sensitive on page	ProUIRadiogroupIsEnabled()	ProUIRadiogroupEnable() ProUIRadiogroupDisable()
.TopOffset on page	ProUIRadiogroupTopoffsetGet()	ProUIRadiogroupTopoffsetSet()
.Visible on page	ProUIRadiogroupIsVisible()	ProUIRadiogroupShow() ProUIRadiogroupHide()

Radiogroup Operations

Functions Introduced

- **ProUIRadiogroupAnchorSet()**
- **ProUIRadiogroupSizeSet()**
- **ProUIRadiogroupMinimumsizeGet()**
- **ProUIRadiogroupPositionSet()**

- **ProUIRadiogroupPositionGet()**
- **ProUIRadiogroupSizeGet()**

Use the function `ProUIRadiogroupAnchorSet()` to set the position of the radiogroup with respect to a given anchor location. This function is applicable only if the parent of the Radiogroup is a drawing area.

Use the function `ProUIRadiogroupSizeSet()` to set the size of the radiogroup. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIRadiogroupMinimumsizeGet()` to retrieve the minimum size of the width and height of the radiogroup in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIRadiogroupPositionSet()` to set the position to the radiogroup with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIRadiogroupPositionGet()` to get the position to the radiogroup with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIRadiogroupSizeGet()` to get the size of the radiogroup. This field is used only if the parent is a drawing area.

Radiogroup Action Callback

Function Introduced:

- **ProUIRadiogroupSelectActionSet()**

Use the function `ProUIRadiogroupSelectActionSet()` to set the select action for a radio group. This function is called when the user selects one of the buttons in the radio group.

Separator

Separator Attributes

Attribute Name	Get Function	Set Function(s)

`.AttachBottom` on page	`ProUISeparatorIsAttachedBottom()`	`ProUISeparatorAttachBottom()`
		
		`ProUISeparatorUnattachBottom`

`.AttachTop` on page	`ProUISeparatorIsAttachedTop()`	`ProUISeparatorAttachTop()`
		
		`ProUISeparatorUnattachTop()`

Attribute Name	Get Function	Set Function(s)
.AttachRight on page	ProUISeparatorIsAttachedRight()	ProUISeparatorAttachRight() ProUISeparatorUnattachRight()
.AttachLeft on page	ProUISeparatorIsAttachedLeft()	ProUISeparatorAttachLeft() ProUISeparatorUnattachLeft()
.BottomOffset on page	ProUISeparatorBottomoffsetGet()	ProUISeparatorBottomoffsetSet()
.LeftOffset on page	ProUISeparatorLeftoffsetGet()	ProUISeparatorLeftoffsetSet()
.PopupMenu on page	ProUISeparatorPopupmenuGet()	ProUISeparatorPopupmenuSet()
.RightOffset on page	ProUISeparatorRightoffsetGet()	ProUISeparatorRightoffsetSet()
.TopOffset on page	ProUISeparatorTopoffsetGet()	ProUISeparatorTopoffsetSet()
.Visible on page	ProUISeparatorIsVisible()	ProUISeparatorShow() ProUISeparatorHide()

Slider

Slider Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUISliderIsAttachedBottom()	ProUISliderAttachBottom() ProUISliderUnattachBottom
.AttachTop on page	ProUISliderIsAttachedTop()	ProUISliderAttachTop() ProUISliderUnattachTop()
.AttachRight on page	ProUISliderIsAttachedRight()	ProUISliderAttachRight() ProUISliderUnattachRight()
.AttachLeft on page	ProUISliderIsAttachedLeft()	ProUISliderAttachLeft() ProUISliderUnattachLeft()
.BottomOffset on page	ProUISliderBottomoffsetGet()	ProUISliderBottomoffsetSet()
.HelpText on page	ProUISliderHelptextGet()	ProUISliderHelptextSet()
.Integer on page	ProUISliderIntegerGet()	ProUISliderIntegerSet()
.LeftOffset on page	ProUISliderLeftoffsetGet()	ProUISliderLeftoffsetSet()
.Length on page	ProUISliderLengthGet()	ProUISliderLengthSet()
.MaxInteger on page	ProUISliderMaxintegerGet()	ProUISliderMaxintegerSet()
.MinInteger on	ProUISliderMinintegerGet()	ProUISliderMinintegerSet()

Attribute Name	Get Function	Set Function(s)
page		
Orientation on page	ProUISliderOrientationGet()	ProUISliderOrientationSet()
.ParentName on page	ProUISliderParentnameGet()	Not Applicable
.PopupMenu on page	ProUISliderPopupmenuGet()	ProUISliderPopupmenuSet()
.RightOffset on page	ProUISliderRightoffsetGet()	ProUISliderRightoffsetSet()
.Sensitive on page	ProUISliderIsEnabled()	ProUISliderEnable() ProUISliderDisable()
.TopOffset on page	ProUISliderTopoffsetGet()	ProUISliderTopoffsetSet()
.Tracking on page	ProUISliderIsTrackingEnabled()	ProUISliderEnableTracking() ProUISliderDisableTracking()
.Visible on page	ProUISliderIsVisible()	ProUISliderShow() ProUISliderHide()

Slider Operations

Functions Introduced

- **ProUISliderAnchorSet()**
- **ProUISliderSizeSet()**
- **ProUISliderMinimumsizeGet()**
- **ProUISliderPositionSet()**
- **ProUISliderPositionGet()**
- **ProUISliderSizeGet()**

Use the function `ProUISliderAnchorSet()` to set the position of the slider with respect to a given anchor location. This function is applicable only if the parent of the Slider is a drawing area.

Use the function `ProUISliderSizeSet()` to set the size of the slider. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUISliderMinimumsizeGet()` to retrieve the minimum size of the width and height of the slider in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUISliderPositionSet()` to set the position to the slider with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUISliderPositionGet()` to get the position to the slider with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUISliderSizeGet()` to get the size of the slider. This field is used only if the parent is a drawing area.

Slider Action Callbacks

Function Introduced

- **ProUISliderUpdateActionSet()**

Use the function `ProUISliderUpdateActionSet()` to set the update action for the slider.

Example 4: Source of Dialog with Slider and Linked InputPanel

The sample code in `UgUISlider.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui` shows code for use of a dialog with Slider and Linked InputPanel.

Spinbox

Spinbox Attributes

Attribute Name	Get Function	Set Function(s)
`.AttachBottom` on page	`ProUISpinboxIsAttachedBottom()`	`ProUISpinboxAttachBottom()` `ProUISpinboxUnattachBottom`
`.AttachTop` on page	`ProUISpinboxIsAttachedTop()`	`ProUISpinboxAttachTop()` `ProUISpinboxUnattachTop()`
`.AttachRight` on page	`ProUISpinboxIsAttachedRight()`	`ProUISpinboxAttachRight()` `ProUISpinboxUnattachRight()`
`.AttachLeft` on page	`ProUISpinboxIsAttachedLeft()`	`ProUISpinboxAttachLeft()` `ProUISpinboxUnattachLeft()`
`.BottomOffset` on page	`ProUISpinboxBottomoffsetGet()`	`ProUISpinboxBottomoffsetSet()`
`.Digits` on page	`ProUISpinboxDigitsGet()`	`ProUISpinboxDigitsSet()`
`.Double` on page	`ProUISpinboxDoubleGet()`	`ProUISpinboxDoubleSet()`
`.DoubleFormat` on page	`ProUISpinboxDoubleformatGet()`	`ProUISpinboxDoubleformatSet()`
`.DoubleIncrement` on page	`ProUISpinboxDoubleincrementGet()`	`ProUISpinboxDoubleincrementSet()`

Attribute Name	Get Function	Set Function(s)
.Editable on page	ProUISpinboxIsEditable()	ProUISpinboxEditable() ProUISpinboxReadOnly()
.FastDoubleIncrement on page	ProUISpinboxFastdoubleincrementGet()	ProUISpinboxFastdoubleincrementSet()
.FastIncrement on page	ProUISpinboxFastincrementGet()	ProUISpinboxFastincrementSet()
.HelpText on page	ProUISpinboxHelptextGet()	ProUISpinboxHelptextSet()
.Increment on page	ProUISpinboxIncrementGet()	ProUISpinboxIncrementSet()
.InputType on page	ProUISpinboxInputtypeGet()	ProUISpinboxInputtypeSet()
.Integer on page	ProUISpinboxIntegerGet()	ProUISpinboxIntegerSet()
.LeftOffset on page	ProUISpinboxLeftoffsetGet()	ProUISpinboxLeftoffsetSet()
.MaxDouble on page	ProUISpinboxMaxdoubleGet()	ProUISpinboxMaxdoubleSet()
.MaxInteger on page	ProUISpinboxMaxintegerGet()	ProUISpinboxMaxintegerSet()
.MinDouble on page	ProUISpinboxMindoubleGet()	ProUISpinboxMindoubleSet()
.MinInteger on page	ProUISpinboxMinintegerGet()	ProUISpinboxMinintegerSet()
.ParentName on page	ProUISpinboxParentnameGet()	Not Applicable
.PopupMenu on page	ProUISpinboxPopupmenuGet()	ProUISpinboxPopupmenuSet()
.RightOffset on page	ProUISpinboxRightoffsetGet()	ProUISpinboxRightoffsetSet()
.Sensitive on page	ProUISpinboxIsEnabled()	ProUISpinboxEnable() ProUISpinboxDisable()
.TopOffset on page	ProUISpinboxTopoffsetGet()	ProUISpinboxTopoffsetSet()
.Visible on page	ProUISpinboxIsVisible()	ProUISpinboxShow() ProUISpinboxHide()

Spinbox Action Callbacks

Functions Introduced:

- **ProUISpinboxUpdateActionSet()**
- **ProUISpinboxActivateActionSet()**

Use the function `ProUISpinboxUpdateActionSet()` and `ProUISpinboxActivateActionSet()` to set the update and activate action for a spin box respectively.

Spinbox Operations

Functions Introduced

- **ProUISpinboxAnchorSet()**
- **ProUISpinboxSizeSet()**
- **ProUISpinboxMinimumsizeGet()**
- **ProUISpinboxPositionSet()**
- **ProUISpinboxPositionGet()**
- **ProUISpinboxSizeGet()**

Use the function `ProUISpinboxAnchorSet()` to set the position of the spinbox with respect to a given anchor location. This function is applicable only if the parent of the Spinbox is a drawing area.

Use the function `ProUISpinboxSizeSet()` to set the size of the spinbox. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUISpinboxMinimumsizeGet()` to retrieve the minimum size of the width and height of the spin box in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUISpinboxPositionSet()` to set the position to the spinbox with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUISpinboxPositionGet()` to get the position to the spinbox with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUISpinboxSizeGet()` to get the size of the spinbox. This field is used only if the parent is a drawing area.

Tab

Tab Attributes

Attribute Name	Get Function	Set Function(s)

`.AttachBottom on page`	`ProUITabIsAttachedBottom()`	`ProUITabAttachBottom()`
		
		`ProUITabUnattachBottom`

`.AttachTop on page`	`ProUITabIsAttachedTop()`	`ProUITabAttachTop()`
		
		`ProUITabUnattachTop()`

`.AttachRight on page`	`ProUITabIsAttachedRight()`	`ProUITabAttachRight()`
		
		`ProUITabUnattachRight()`

Attribute Name	Get Function	Set Function(s)
.AttachLeft on page	ProUITabIsAttached Left()	ProUITabAttachLeft() ProUITabUnattachLeft()
.BottomOffset on page	ProUITabBottomoffset Get()	ProUITabBottomoffsetSet()
.Decorated on page	ProUITabIsDecorated()	ProUITabDecorate() ProUITabUndecorate()
.HelpText on page	ProUITabHelptextGet()	ProUITabHelptextSet()
.ItemHelpText on page	ProUITabItemhelptext Get()	ProUITabItemhelptextSet()
.Label on page	ProUITabLabelsGet()	ProUITabLabelsSet()
.LeftOffset on page	ProUITabLeftoffset Get()	ProUITabLeftoffsetSet()
.ParentName on page	ProUITabParentname Get()	Not Applicable
.PopupMenu on page	ProUITabPopupmenuGet()	ProUITabPopupmenuSet()
.RightOffset on page	ProUITabRightoffset Get()	ProUITabRightoffsetSet()
.SelectedNames on page	ProUITabSelectednames Get()	ProUITabSelectednamesSet()
.Sensitive on page	ProUITabIsEnabled()	ProUITabEnable() ProUITabDisable()
.TopOffset on page	ProUITabTopoffsetGet()	ProUITabTopoffsetSet()
.Visible on page	ProUITabIsVisible()	ProUITabShow() ProUITabHide()

Tab Operations

Functions Introduced

- **ProUITabAnchorSet()**
- **ProUITabSizeSet()**
- **ProUITabMinimumsizeGet()**
- **ProUITabPositionSet()**
- **ProUITabPositionGet()**
- **ProUITabSizeGet()**
- **ProUITabLayoutAdd()**
- **ProUITabLayoutsInsert()**
- **ProUITabItemNameSet()**

- **ProUITabItemLabelSet()**
- **ProUITabItemImageSet()**
- **ProUITabItemHelptextStringSet()**
- **ProUITabItemLabelGet()**
- **ProUITabItemImageGet()**
- **ProUITabItemHelptextStringGet()**
- **ProUITabItemExtentsGet()**

Use the function `ProUITabAnchorSet()` to set the position of the Tab with respect to a given anchor location. This function is applicable only if the parent of the Tab is a drawing area.

Use the function `ProUITabSizeSet()` to set the size of the Tab. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUITabMinimumsizeGet()` to retrieve the minimum size of the width and height of the tab in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUITabPositionSet()` to set the position to the Tab with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUITabPositionGet()` to get the position to the Tab with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUITabSizeGet()` to get the size of the Tab. This field is used only if the parent is a drawing area.

Use the function `ProUITabLayoutAdd()` to add a new layout to the tab. Even if the layout has a label assigned, you must set the tab labels with `ProUITabLabelsSet()` for the decorated tab to show the appropriate labels.

Use the function `ProUITabLayoutsInsert()` to insert a new layout after an existing component in the tab. Even if the layouts have labels assigned, you must set the tab labels with `ProUITabLabelsSet()` for the decorated tab to show the appropriate labels.

Use the function `ProUITabItemNameSet()` to set a new name to the item in the tab.

Use the function `ProUITabItemLabelSet()` to set the label of the item in the tab.

Use the function `ProUITabItemImageSet()` to set the image of the item in the tab.

Use the function `ProUITabItemHelptextStringSet()` to set the text that should be displayed when the cursor is over the item in the tab.

Use the function `ProUITabItemLabelGet()` to get the label of the item in the tab.

Use the function `ProUITabItemImageGet()` to get the image of the item in the tab.

Use the function `ProUITabItemHelptextStringGet()` to get the text that is displayed when the cursor is over the item in the tab.

Use the function `ProUITabItemExtentsGet()` to get the boundary of the item that is in the tab relative to the top left corner of the dialog.

Tab Action Callbacks

Function Introduced

- **ProUITabSelectActionSet()**

Use the function `ProUITabSelectActionSet()` sets the select action for a tab. This function is called when the user selects the tab.

Table

Table Attributes

Attribute Name	Get Function	Set Function(s)
`.ActivateOnReturn` on page	`ProUITableIsActivateonreturnEnabled()`	`ProUITableActivateonreturnEnable()` `ProUITableActivateonreturnDisable()`
`.Alignment` on page	`ProUITableAlignmentGet()`	`ProUITableAlignmentSet()`
`.AttachBottom` on page	`ProUITableIsAttachedBottom()`	`ProUITableAttachBottom()` `ProUITableUnattachBottom`
`.AttachTop` on page	`ProUITableIsAttachedTop()`	`ProUITableAttachTop()` `ProUITableUnattachTop()`
`.AttachRight` on page	`ProUITableIsAttachedRight()`	`ProUITableAttachRight()` `ProUITableUnattachRight()`
`.AttachLeft` on page	`ProUITableIsAttachedLeft()`	`ProUITableAttachLeft()` `ProUITableUnattachLeft()`
`.Autohighlight` on page	`ProUITableIsAutohighlightEnabled()`	`ProUITableAutohighlightEnable()` `ProUITableAutohighlightDisable()`
`.BottomOffset` on page	`ProUITableBottomoffsetGet()`	`ProUITableBottomoffsetSet()`

Attribute Name	Get Function	Set Function(s)
.ChildNames on page	ProUITableChildnamesGet()	Not Applicable
.ColumnLabels on page	ProUITableColumnlabelsGet()	ProUITableColumnlabelsSet()
.ColumnNames on page	ProUITableColumnnamesGet()	ProUITableColumnnamesSet()
.ColumnWidths on page	ProUITableColumnwidthsGet()	ProUITableColumnwidthsSet()
.ColumnResizings on page	ProUITableColumnresizingsGet()	ProUITableColumnresizingsSet()
.ColumnSelectionPolicy on page	ProUITableColumnselectionpolicyGet()	ProUITableColumnselectionpolicySet()
.Columns on page	ProUITableColumnsGet()	ProUITableColumnsSet()
.DefaultColumnWidth on page	ProUITableDefaultcolumnwidthGet()	ProUITableDefaultcolumnwidthSet()
.HelpText on page	ProUITableHelptextGet()	ProUITableHelptextSet()
.LeftOffset on page	ProUITableLeftoffsetGet()	ProUITableLeftoffsetSet()
.LockedColumns on page	ProUITableLockedcolumnsGet()	ProUITableLockedcolumnsSet()
.LockedRows on page	ProUITableLockedrowsGet()	ProUITableLockedrowsSet()
.MinColumns on page	ProUITableMincolumnsGet()	ProUITableMincolumnsSet()
.MinRows on page	ProUITableMinrowsGet()	ProUITableMinrowsSet()
.ParentName on page	ProUITableParentnameGet()	Not Applicable
.PopupMenu on page	ProUITablePopupmenuGet()	ProUITablePopupmenuSet()
.RightOffset on page	ProUITableRightoffsetGet()	ProUITableRightoffsetSet()
.RowLabels on page	ProUITableRowlabelsGet()	ProUITableRowlabelsSet()
.RowNames on page	ProUITableRownamesGet()	ProUITableRownamesSet()
.RowSelectionPolicy on page	Not Applicable	Not Applicable
.ScrollBarsWhenNeeded on page	ProUITableUsesScrollbarswhenneeded()	ProUITableUseScrollbarswhenNeeded() ProUITableAlwaysUsescrollBars()
.SelectedNames on page	ProUITableSelectednamesGet()	ProUITableSelectednamesSet()
.SelectedColumnNames on page	ProUITableSelectedcolumnnamesGet()	ProUITableSelectedcolumnnamesSet()
.SelectedRowNames on page	ProUITableSelectedrownamesGet()	ProUITableSelectedrownamesSet()
.Sensitive on page	ProUITableIsEnabled()	ProUITableEnable() ProUITableDisable()
.SelectionPolicy on page	ProUITableSelectionpolicyGet()	ProUITableSelectionpolicySet()

Attribute Name	Get Function	Set Function(s)
.ShowGrid on page	ProUITableShowgridGet()	ProUITableShowgridSet()
.TopOffset on page	ProUITableTopoffsetGet()	ProUITableTopoffsetSet()
.TruncateLabel on page	ProUITableTruncatelabelGet()	ProUITableTruncatelabelSet()
.Visible on page	ProUITableIsVisible()	ProUITableShow() ProUITableHide()
.VisibleRows on page	ProUITableVisiblerowsGet()	ProUITableVisiblerowsSet()

Adding and Removing Components

Component Name	Adding Functions	Removing Functions
Checkbutton	ProUITableCheckbuttonAdd()	ProUITableCheckbuttonRemove()
Drawingarea	ProUITableDrawingareaAdd()	ProUITableDrawingareaRemove()
Inputpanel	ProUITableInputpanelAdd()	ProUITableInputpanelRemove()
Label	ProUITableLabelAdd()	ProUITableLabelRemove()
Layout	ProUITableLayoutAdd()	ProUITableLayoutRemove()
List	ProUITableListAdd()	ProUITableListRemove()
Optionmenu	ProUITableOptionmenuAdd()	ProUITableOptionmenuRemove()
Progressbar	ProUITableProgressbarAdd()	ProUITableProgressbarRemove()
Pushbutton	ProUITablePushbuttonAdd()	ProUITablePushbuttonRemove()
Radiogroup	ProUITableRadiogroupAdd()	ProUITableRadiogroupRemove()
Slider	ProUITableSliderAdd()	ProUITableSliderRemove()
Spinbox	ProUITableSpinboxAdd()	ProUITableSpinboxRemove()
Tab	ProUITableTabAdd()	ProUITableTabRemove()
Table	ProUITableTableAdd()	ProUITableTableRemove()
Textarea	ProUITableTextareaAdd()	ProUITableTextareaRemove()
Thumbwheel	ProUITableThumbwheelAdd()	ProUITableThumbwheelRemove()
Tree	ProUITableTreeAdd()	ProUITableTreeRemove()

Table Cell Functions

Functions Introduced

- **ProUITableCellLabelSet()**
- **ProUITableCellLabelGet()**
- **ProUITableCellHelpTextSet()**

- **ProUITableIsCellSensitive()**
- **ProUITableCellEnable()**
- **ProUITableCellDisable()**
- **ProUITableCellComponentNameSet()**
- **ProUITableCellComponentNameGet()**
- **ProUITableCellComponentCopy()**
- **ProUITableCellComponentDelete()**
- **ProUITableAnchorCellSet()**
- **ProUITableAnchorCellGet()**
- **ProUITableFocusCellSet()**
- **ProUITableFocusCellGet()**
- **ProUITableCellHelptextStringGet()**
- **ProUITableCellHelptextStringSet()**
- **ProUITableCellForegroundColorGet()**
- **ProUITableCellForegroundColorSet()**
- **ProUITableCellBackgroundColorGet()**
- **ProUITableCellBackgroundColorSet()**

Use the function `ProUITableCellLabelSet()` to set the text contained in the table cell.

Use the function `ProUITableCellLabelGet()` to get the text contained in the table cell.

Use the function `ProUITableCellHelpTextSet()` to set the help text for the table cell.

Use the function `ProUITableIsCellSensitive()` to determine if the table cell is sensitive to user input.

Use the function `ProUITableCellEnable()` to set the table cell to be sensitive to user input.

Use the function `ProUITableCellDisable()` to set the table cell to be insensitive to user input.

Use the function `ProUITableCellComponentNameSet()` to set the component name contained in the table cell.

Use the function `ProUITableCellComponentNameGet()` to get the component name contained in the table cell.

Use the function `ProUITableCellComponentCopy()` to copy a predefined component and places it in the table in the designated cell. The component will be displayed in this cell.

Use the function `ProUITableCellComponentDelete()` to remove the component contained in the table cell.

Use the function `ProUITableAnchorCellSet()` to set the coordinates of the table selection anchor cell.

Use the function `ProUITableAnchorCellGet()` to get the coordinates of the table selection anchor cell.

Use the function `ProUITableFocusCellSet()` to set the coordinates of the table selection focus cell.

Use the function `ProUITableFocusCellGet()` to get the coordinates of the table selection focus cell.

Use the function `ProUITableCellHelptextStringGet()` to get the help-text that is displayed while the cursor is over the table cell.

Use the function `ProUITableCellHelptextStringSet()` to set the help-text to be displayed when the cursor is over the table cell.

Use the function `ProUITableCellForegroundColorGet()` to get the foreground color of the table cell.

Use the function `ProUITableCellForegroundColorSet()` to set the foreground color of the table.

Use the function `ProUITableCellBackgroundColorGet()` to get the background color of the table cell.

Use the function `ProUITableCellBackgroundColorSet()` to set the background color of the table cell.

Example 5: To Assign Components into Table Cells

The sample code in `UgUITables.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui` shows how to assign components into table cells.

Table Row Functions

Functions Introduced

- **ProUITableRowIndexGet()**
- **ProUITableRowRename()**
- **ProUITableRowLabelSet()**
- **ProUITableRowLabelGet()**
- **ProUITableRowsInsert()**
- **ProUITableRowsDelete()**

- **ProUITableRowCellsSelect()**
- **ProUITableRowCellsDeselect()**
- **ProUITableRowCellLabelsSet()**

Use the function `ProUITableRowIndexGet()` to get the index of the table row with the given name.

Use the function `ProUITableRowRename()` to rename the table row.

Use the function `ProUITableRowLabelSet()` to set the user-visible label for the table row.

Use the function `ProUITableRowLabelGet()` to get the user-visible label for the table row.

Use the function `ProUITableRowsInsert()` to insert one or more new rows into the table.

Use the function `ProUITableRowsDelete()` to delete one or more rows from the table.

Use the function `ProUITableRowCellsSelect()` to select the cells of the given rows of the table. The table selection policy must be either Multiple or Extended.

Use the function `ProUITableRowCellsDeselect()` to deselect the cells of the given rows of the table.

Use the function `ProUITableRowCellLabelsSet()` to set the contents of the cells of the rows using a single string.

Table Column Functions

Functions Introduced

- **ProUITableColumnIndexGet()**
- **ProUITableColumnRename()**
- **ProUITableColumnLabelSet()**
- **ProUITableColumnLabelGet()**
- **ProUITableColumnWidthSet()**
- **ProUITableColumnWidthGet()**
- **ProUITableColumnResizingFactorSet()**
- **ProUITableColumnResizingFactorGet()**
- **ProUITableColumnsInsert()**
- **ProUITableColumnsDelete()**
- **ProUITableColumnCellsSelect()**

- **ProUITableColumnCellsDeselect()**
- **ProUITableResetColumnWidth()**

Use the function `ProUITableColumnIndexGet()` to get the column index for a given column.

Use the function `ProUITableColumnRename()` to rename a table column.

Use the function `ProUITableColumnLabelSet()` to set the user visible label for the column.

Use the function `ProUITableColumnLabelGet()` to get the user-visible label for the column.

Use the function `ProUITableColumnWidthSet()` to set the width of the column in the table.

Use the function `ProUITableColumnWidthGet()` to get the width of the column in the table.

Use the function `ProUITableColumnResizingFactorSet()` to set the resizing factor of the column in the table.

Use the function `ProUITableColumnResizingFactorGet()` to get the resizing factor of the column in the table.

Use the function `ProUITableColumnsInsert()` to insert one or more columns into the table.

Use the function `ProUITableColumnsDelete()` to delete one or more columns from the table.

Use the function `ProUITableColumnCellsSelect()` to select the cells of the given columns in the table.

Use the function `ProUITableColumnCellsDeselect()` to deselect the cells of the given columns in the table.

Use the function `ProUITableResetColumnWidth()` to set the column width to the default.

Example 6: To Access and Modify Names and Labels for the Table Rows and Columns

The sample code in `UgUITables.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui` shows how to access and modify names and labels for the Table rows and columns and also to access, allocate and free the string and wide string arrays.

Table Operations

Functions Introduced

- **ProUITableAnchorSet()**
- **ProUITableSizeSet()**
- **ProUITableMinimumsizeGet()**
- **ProUITablePositionSet()**
- **ProUITablePositionGet()**
- **ProUITableSizeGet()**
- **ProUITableComponentCopy()**
- **ProUITableComponentDelete()**

Use the function `ProUITableAnchorSet()` to set the position of the table with respect to a given anchor location. This function is applicable only if the parent of the table is a drawing area.

Use the function `ProUITableSizeSet()` to set the size of the table. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUITableMinimumsizeGet()` to retrieve the minimum size of the width and height of the table in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUITablePositionSet()` to set the position to the table with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUITablePositionGet()` to get the position to the table with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUITableSizeGet()` to get the size of the table. This field is used only if the parent is a drawing area.

Use the function `ProUITableComponentCopy()` to copy a predefined component and place it in the table. The component is not displayed until it is assigned to a table cell using `ProUITableCellComponentNameSet()`. However, you can update the component's properties as needed and display it at a later time.

Use the function `ProUITableComponentDelete()` to delete a specified component from the table.

Table Action Callbacks

Functions Introduced

- **ProUITableArmActionSet()**
- **ProUITableDisarmActionSet()**
- **ProUITableSelectActionSet()**
- **ProUITableActivateActionSet()**

- **ProUITableFocusinActionSet()**
- **ProUITableFocusoutActionSet()**
- **ProUITableColumnselectActionSet()**

Use the function `ProUITableArmActionSet()` to set the arm action for a table. This function is called when the user changes the selection anchor cell and focus cell in the table.

Use the function `ProUITableDisarmActionSet()` to set the disarm action for a table. This function is called when the user changes the selection focus cell in the table.

Use the function `ProUITableSelectActionSet()` to set the select action for a table. This function is called when the user changes the selected cells in the table.

Use the function `ProUITableActivateActionSet()` to set the activate action for a table. This function is called when the user presses the return key or double-clicks the left mouse button in the table.

Use the function `ProUITableFocusinActionSet()` to set the focus in action for a table.

Use the function `ProUITableFocusoutActionSet()` to set the focus out action for a table.

Use the function `ProUITableColumnselectActionSet()` to set the column selection action for the table. This function is called when the user changes the currently selected table columns.

Example 7: To Access Selected Names Array from Tables

The sample code in `UgUITables.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ui` shows how to access selected names array from tables.

Textarea

Textarea Attributes

Attribute Name	Get Function	Set Function(s)

`.AttachBottom` on page	`ProUITextareaIsAttachedBottom()`	`ProUITextareaAttachBottom()`
		
		`ProUITextareaUnattachBottom()`

`.AttachTop` on page	`ProUITextareaIsAttachedTop()`	`ProUITextareaAttachTop()`
		
		`ProUITextareaUnattachTop()`

Attribute Name	Get Function	Set Function(s)
.AttachRight on page	ProUITextareaIsAttachedRight()	ProUITextareaAttachRight() ProUITextareaUnattachRight()
.AttachLeft on page	ProUITextareaIsAttachedLeft()	ProUITextareaAttachLeft() ProUITextareaUnattachLeft()
.BottomOffset on page	ProUITextareaBottomoffsetGet()	ProUITextareaBottomoffsetSet()
.Columns on page	ProUITextareaColumnsGet()	ProUITextareaColumnsSet()
.Editable on page	ProUITextareaIsEditable()	ProUITextareaEditable() ProUITextareaReadOnly()
.HelpText on page	ProUITextareaHelptextGet()	ProUITextareaHelptextSet()
.LeftOffset on page	ProUITextareaLeftoffsetGet()	ProUITextareaLeftoffsetSet()
.MinRows on page	ProUITextareaMinrowsGet()	ProUITextareaMinrowsSet()
.MaxLen on page	ProUITextareaMaxlenGet()	ProUITextareaMaxlenSet()
.ParentName on page	ProUITextareaParentnameGet()	Not Applicable
.PopupMenu on page	ProUITextareaPopupmenuGet()	ProUITextareaPopupmenuSet()
.RightOffset on page	ProUITextareaRightoffsetGet()	ProUITextareaRightoffsetSet()
.Rows on page	ProUITextareaRowsGet()	ProUITextareaRowsSet()
.Sensitive on page	ProUITextareaIsEnabled()	ProUITextareaEnable() ProUITextareaDisable()
.TopOffset on page	ProUITextareaTopoffsetGet()	ProUITextareaTopoffsetSet()
.Value on page	ProUITextareaValueGet()	ProUITextareaValueSet()
.Visible on page	ProUITextareaIsVisible()	ProUITextareaShow() ProUITextareaHide()

Textarea Operations

Functions Introduced

- **ProUITextareaAnchorSet()**
- **ProUITextareaSizeSet()**
- **ProUITextareaMinimumsizeGet**
- **ProUITextareaPositionGet()**
- **ProUITextareaPositionSet()**
- **ProUITextareaSizeGet()**

Use the function `ProUITextareaAnchorSet()` to set the position of the Textarea with respect to a given anchor location. This function is applicable only if the parent of the Textarea is a drawing area.

Use the function `ProUITextareaSizeSet()` to set the size of the Textarea. This operation is applicable only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUITextareaMinimumsizeGet` to retrieve the minimum size of the width and height of the textarea in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUITextareaPositionGet()` to get the position of the Textarea with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUITextareaPositionSet()` to set the position to the Textarea with respect to its parent. This operation is applicable only if the parent is a drawing area.

Use the function `ProUITextareaSizeGet()` to get the size of the Textarea. This operation is applicable only if the parent is a drawing area.

Textarea Action Callbacks

Functions Introduced

- **ProUITextareaActivateActionSet()**
- **ProUITextareaFocusinActionSet()**
- **ProUITextareaFocusoutActionSet()**
- **ProUITextareaInputActionSet()**

Use the function `ProUITextareaActivateActionSet()` to set the action callback to be called when the user hits return in an Textarea.

Use the function `ProUITextareaFocusinActionSet()` to set the focus in action for an Textarea. This function is called when the user moves the cursor onto the Textarea using the mouse or `<TAB>` key.

Use the function `ProUITextareaFocusoutActionSet()` to set the focus out action for an Textarea. This function is called when the user moves the cursor off of the Textarea using the mouse or `<TAB>` key.

Use the function `ProUITextareaInputActionSet()` to set the action callback to be called when the user enters a key in an Textarea.

Thumbwheel

Thumbwheel Attributes

Attribute Name	Get Function	Set Function(s)
.AttachBottom on page	ProUIThumbwheelIsAttachedBottom()	ProUIThumbwheelAttachBottom() ProUIThumbwheelUnattachBottom
.AttachTop on page	ProUIThumbwheelIsAttachedTop()	ProUIThumbwheelAttachTop() ProUIThumbwheelUnattachTop()
.AttachRight on page	ProUIThumbwheelIsAttachedRight()	ProUIThumbwheelAttachRight() ProUIThumbwheelUnattachRight()
.AttachLeft on page	ProUIThumbwheelIsAttachedLeft()	ProUIThumbwheelAttachLeft() ProUIThumbwheelUnattachLeft()
.BottomOffset on page	ProUIThumbwheelBottomoffsetGet()	ProUIThumbwheelBottomoffsetSet()
.HelpText on page	ProUIThumbwheelHelptextGet()	ProUIThumbwheelHelptextSet()
.Integer on page	ProUIThumbwheelIntegerGet()	ProUIThumbwheelIntegerSet()
.LeftOffset on page	ProUIThumbwheelLeftoffsetGet()	ProUIThumbwheelLeftoffsetSet()
.MaxInteger on page	ProUIThumbwheelMaxintegerGet()	ProUIThumbwheelMaxintegerSet()
.MinInteger on page	ProUIThumbwheelMinintegerGet()	ProUIThumbwheelMininteger Set()
.ParentName on page	ProUIThumbwheelParentnameGet()	Not Applicable
.PopupMenu on page	ProUIThumbwheelPopupmenuGet()	ProUIThumbwheelPopupmenuSet()
.RightOffset on page	ProUIThumbwheelRightoffsetGet()	ProUIThumbwheelRightoffsetSet()
.Sensitive on page	ProUIThumbwheelIsEnabled()	ProUIThumbwheelEnable() ProUIThumbwheelDisable()
.TopOffset on page	ProUIThumbwheelTopoffsetGet()	ProUIThumbwheelTopoffsetSet()
.UnitsPerRotation on page	ProUIThumbwheelUnitsperrotationGet()	ProUIThumbwheelUnitsperrotationSet()
.Visible on page	ProUIThumbwheelIsVisible()	ProUIThumbwheelShow() ProUIThumbwheelHide()

Thumbwheel Operations

Functions Introduced

- **ProUIThumbwheelAnchorSet()**
- **ProUIThumbwheelSizeSet()**
- **ProUIThumbwheelMinimumsizeGet()**
- **ProUIThumbwheelPositionSet()**
- **ProUIThumbwheelPositionGet()**
- **ProUIThumbwheelSizeGet()**

Use the function `ProUIThumbwheelAnchorSet()` to set the position of the Thumbwheel with respect to a given anchor location. This function is applicable only if the parent of the Thumbwheel is a drawing area.

Use the function `ProUIThumbwheelSizeSet()` to set the size of the Thumbwheel. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUIThumbwheelMinimumsizeGet()` to retrieve the minimum size of the width and height of the thumb wheel in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUIThumbwheelPositionSet()` to set the position to the Thumbwheel with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIThumbwheelPositionGet()` to get the position to the Thumbwheel with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUIThumbwheelSizeGet()` to get the size of the Thumbwheel. This field is used only if the parent is a drawing area.

Thumbwheel Action Callbacks

Functions Introduced

- **ProUIThumbwheelUpdateActionSet()**

Use the function `ProUIThumbwheelUpdateActionSet()` to set the update action for the Thumbwheel.

Tree

Tree Attributes

Attribute Name	Get Function	Set Function(s)
.ActivateOnReturn on page	ProUITreeIsActivateonreturnEnabled()	ProUITreeActivateonreturnEnable() ProUITreeActivateonreturnDisable()
.AttachBottom on page	ProUITreeIsAttachedBottom()	ProUITreeAttachBottom() ProUITreeUnattachBottom()
.AttachLeft on page	ProUITreeIsAttachedLeft()	ProUITreeAttachLeft() ProUITreeUnattachLeft()
.AttachRight on page	ProUITreeIsAttachedRight()	ProUITreeAttachRight() ProUITreeUnattachRight()
.AttachTop on page	ProUITreeIsAttachedTop()	ProUITreeAttachTop() ProUITreeUnattachTop()
.BackgroundColor on page	ProUITreeBackgroundcolorGet()	ProUITreeBackgroundcolorSet()
.BottomOffset on page	ProUITreeBottomoffsetGet()	ProUITreeBottomoffsetSet()
.LeftOffset on page	ProUITreeLeftoffsetGet()	ProUITreeLeftoffsetSet()
.RightOffset on page	ProUITreeRightoffsetGet()	ProUITreeRightoffsetSet()
.TopOffset on page	ProUITreeTopoffsetGet()	ProUITreeTopoffsetSet()
.Columns on page	Not Applicable	Not Applicable
.ForegroundColor on page	ProUITreeForegroundcolorGet()	ProUITreeForegroundcolorSet()
.HelpText on page	ProUITreeHelptextGet()	ProUITreeHelptextSet()
.Label on page	Not Applicable	Not Applicable
.LabelAlignment on page	Not Applicable	Not Applicable
.Lastentereditem on page	ProUITreeLastentereditemGet()	Not Applicable
.Mapped on page	ProUITreeIsMapped()	ProUITreeMappedSet() ProUITreeMappedUnset()
.MinColumns on page	Not Applicable	Not Applicable
.MinRows on page	Not Applicable	Not Applicable
.MixedState on page	ProUITreeMixedStateGet()	ProUITreeMixedStateSet()
.ParentName on page	ProUITreeParentnameGet()	Not Applicable

Attribute Name	Get Function	Set Function(s)
.PopupMenu on page	ProUITreePopupmenuGet()	ProUITreePopupmenuSet()
.PopupWhenInsen on page	ProUITreeIsPopupwheninsensitiveEnabled()	ProUITreeEnablePopupwheninsensitive() ProUITreeDisablePopupwheninsensitive()
.ResizableCols on page	Not Applicable	Not Applicable
.Rows on page	ProUITreeRowsGet()	ProUITreeRowsSet()
.ScrollBarsWhen Needed on page	ProUITreeUsesScrollbarswhenneeded()	ProUITreeUseScrollbarswhenNeeded() ProUITreeAlwaysUsescrollBars()
.SelectedNames on page	ProUITreeSelectednamesGet()	ProUITreeSelectednamesSet()
.SelectionPolicy on page	ProUITreeSelectionpolicyGet()	ProUITreeSelectionpolicySet()
.SelectByCell on page	Not Applicable	Not Applicable
.SelectCBRegardless on page	ProUITreeSelectcbregardlessGet()	ProUITreeSelectcbregardlessSet()
.Sensitive on page	ProUITreeIsEnabled()	ProUITreeEnable() ProUITreeDisable()
.TreeAttributeWindow on page	Not Applicable	Not Applicable
.TreeBoxNodes on page	Not Applicable	Not Applicable
.TreeCellInput on page	ProUITreeTreecellinputGet()	Not Applicable
.TreeCellSelCol on page	ProUITreeTreecellselcolGet()	ProUITreeTreecellselcolSet()
.TreeCellSelNode on page	ProUITreeTreecellselnodeGet()	ProUITreeTreecellselnodeSet()
.TreeChildOrder on page	Not Applicable	Not Applicable
.TreeColumnsJusts on page	Not Applicable	Not Applicable
.TreeColumnsNames on page	Not Applicable	Not Applicable
.TreeColumnsTitles on page	Not Applicable	Not Applicable
.TreeColumnsWidths on page	Not Applicable	Not Applicable
.TreeColumnOrder on page	ProUITreeTreecolumnorderGet()	ProUITreeTreecolumnorderSet()
.TreeCurrent on page	ProUITreeTreecurrentnodeGet()	ProUITreeTreecurrentnodeSet()
.TreeDisplayRoot	ProUITreeIsRootnodeVisi	ProUITreeShowRootnode()

Attribute Name	Get Function	Set Function(s)
on page	ble()	ProUITreeHideRootnode()
.TreeExpColNode on page	ProUITreeTreeexpcolnode Get()	Not Applicable
.TreeIndicate Children on page	Not Applicable	Not Applicable
.TreeInitialSash Percent on page	Not Applicable	Not Applicable
.TreeKeyboardIn put on page	ProUITreeIsCellkeyboardin putEnabled()	ProUITreeEnableCellkeyboar dinput() ProUITreeDisableCellkeyboar dinput()
.TreeLinkStyle on page	Not Applicable	Not Applicable
.TreeNodeTypeAp pends on page	ProUITreeTreenodetypeap pendsGet()	ProUITreeTreenodetypeap pendsSet()
.TreeNodeTypeCol lapsedImages on page	ProUITreeTreenodetypecol lapsedimagesGet()	ProUITreeTreenodetypecollap sedimagesSet()
.TreeNodeTypeEx pandedImages on page	ProUITreeTreenodetypeexpan dedimagesGet()	ProUITreeTreenodetypeexpan dedimagesSet()
.TreeNodeType HelpTexts on page	ProUITreeTreenodetypehelp textsGet()	ProUITreeTreenodetypehelp textsSet()
.TreeNodeTypePre fixs on page	ProUITreeTreenodetypepre fixesGet	ProUITreeTreenodetypeprefix esSet()
.TreeNodeType Names on page	ProUITreeTreenodetypenames Get()	ProUITreeTreenodetypenames Set()
.TreeRedraw on page	ProUITreeTreeredrawGet()	ProUITreeTreeredrawSet()
.TreeRootNode on page	ProUITreeTreerootnodeGet()	ProUITreeTreerootnodeSet()
.TreeState on page	ProUITreeStateGet()	ProUITreeStateSet()
.TreeVerticalSB Position on page	Not Applicable	Not Applicable
.Visible on page	ProUITreeIsVisible()	ProUITreeShow() ProUITreeHide()

Note

Many of the properties of trees are currently only supported as a part of the resource file. Only very basic trees can be created using programmatic means.

Adding and Removing Components

Component Name	Adding Functions	Removing Functions
Checkbutton	`ProUITreeCheckbuttonAdd()`	`ProUITreeCheckbuttonRemove()`
Drawingarea	`ProUITreeDrawingareaAdd()`	`ProUITreeDrawingareaRemove()`
Inputpanel	`ProUITreeInputpanelAdd()`	`ProUITreeInputpanelRemove()`
Label	`ProUITreeLabelAdd()`	`ProUITreeLabelRemove()`
Layout	`ProUITreeLayoutAdd()`	`ProUITreeLayoutRemove()`
List	`ProUITreeListAdd()`	`ProUITreeListRemove()`
Optionmenu	`ProUITreeOptionmenuAdd()`	`ProUITreeOptionmenuRemove()`
Progressbar	`ProUITreeProgressbarAdd()`	`ProUITreeProgressbarRemove()`
Pushbutton	`ProUITreePushbuttonAdd()`	`ProUITreePushbuttonRemove()`
Radiogroup	`ProUITreeRadiogroupAdd()`	`ProUITreeRadiogroupRemove()`
Slider	`ProUITreeSliderAdd()`	`ProUITreeSliderRemove()`
Spinbox	`ProUITreeSpinboxAdd()`	`ProUITreeSpinboxRemove()`
Tab	`ProUITreeTabAdd()`	`ProUITreeTabRemove()`
Table	`ProUITreeTableAdd()`	`ProUITreeTableRemove()`
Tree	`ProUITreeTreeAdd()`	`ProUITreeTreeRemove()`
Textarea	`ProUITreeTextareaAdd()`	`ProUITreeTextareaRemove()`
Thumbwheel	`ProUITreeThumbwheelAdd()`	`ProUITreeThumbwheelRemove()`

Tree Column Functions

Functions Introduced

- **ProUITreeColumnCreate()**
- **ProUITreeColumnTitleGet()**
- **ProUITreeColumnWidthGet()**
- **ProUITreeColumnVisibilityGet()**
- **ProUITreeColumnJustificationGet()**
- **ProUITreeColumnTitleSet()**
- **ProUITreeColumnWidthSet()**
- **ProUITreeColumnVisibilitySet()**
- **ProUITreeColumnJustificationSet()**

Use the function `ProUITreeColumnCreate()` to create a column of the given name in the attribute window of the tree. The name must be unique within the scope of the columns of the tree.

Use the function `ProUITreeColumnTitleGet()` to obtain the title of a column in the tree.

Use the function `ProUITreeColumnWidthGet()` to obtain the width of a column in the tree.

Use the function `ProUITreeColumnVisibilityGet()` to obtain the visibility of a column in the tree.

Use the function `ProUITreeColumnJustificationGet()` to obtain the justification of a column in the tree.

Use the function `ProUITreeColumnTitleSet()` to set a title to the column in the tree.

Use the function `ProUITreeColumnWidthSet()` to set a width to the column in the tree.

Use the function `ProUITreeColumnVisibilitySet()` to set visibility of the column in the tree.

Use the function `ProUITreeColumnJustificationSet()` to set justification to the column in the tree.

Tree Node Functions

Functions Introduced

- **ProUITreeNodeComponentSet()**
- **ProUITreeNodeChildrenGet()**
- **ProUITreeNodeLabelGet()**
- **ProUITreeNodeLabelSet()**
- **ProUITreeNodeParentGet()**
- **ProUITreeNodeTypeGet()**
- **ProUITreeNodeTypeSet()**
- **ProUITreeNodeExtentsGet()**
- **ProUITreeNodesOfTypeGet()**
- **ProUITreeNodeAdd()**
- **ProUITreeNodeInsert()**
- **ProUITreeNodeCollapse()**
- **ProUITreeNodeDelete()**

- **ProUITreeNodeExists()**
- **ProUITreeNodeExpand()**
- **ProUITreeNodeHelptextGet()**
- **ProUITreeNodeIsVisible()**
- **ProUITreeNodeIsExpanded()**
- **ProUITreeNodeIsSelected()**
- **ProUITreeNodeFontstyleGet()**
- **ProUITreeNodeHelptextSet()**
- **ProUITreeNodeShow()**
- **ProUITreeNodeHide()**
- **ProUITreeNodeSelect()**
- **ProUITreeNodeDeselect()**
- **ProUITreeNodeFontstyleSet()**
- **ProUITreeNodeMove()**
- **ProUITreeNodeRename()**
- **ProUITreeNodetypeAdd()**
- **ProUITreeNodetypeDelete()**
- **ProUITreeNodeColumntextSet()**
- **ProUITreeNodesSelect()**
- **ProUITreeNodetypeSelect()**
- **ProUITreeAllnodesSelect()**
- **ProUITreeNodesDeselect()**
- **ProUITreeNodetypeDeselect()**
- **ProUITreeAllnodesDeselect()**
- **ProUITreeNodeVisitAction()**
- **ProUITreeNodeFilterAction()**
- **ProUITreeNodesVisit()**

Use the function `ProUITreeNodeComponentSet()` to set the component to be displayed in the given column corresponding to the given node name of the tree.

Use the function `ProUITreeNodeChildrenGet()` to get the child nodes for the node in the tree.

Use the function `ProUITreeNodeLabelGet()` to get the label of the node in the tree.

Use the function `ProUITreeNodeLabelSet()` to set the label of the node in the tree.

Use the function `ProUITreeNodeParentGet()` to get the parent of the node in the tree.

Use the function `ProUITreeNodeTypeGet()` to get the name of the node type associated with the node.

Use the function `ProUITreeNodeTypeSet()` to set the name of the node type associated with the node.

Use the function `ProUITreeNodeExtentsGet()` to get the boundary of the node of the tree relative to the top-left corner of the dialog.

Use the function `ProUITreeNodesOfTypeGet()` to get the name of all nodes of the tree associated with the node type.

Use the function `ProUITreeNodeAdd()` to add a new node to the tree.

Use the function `ProUITreeNodeInsert()` to insert a new node before a node in the tree.

Use the function `ProUITreeNodeCollapse()` to collapse a node in the tree making its children invisible.

Use the function `ProUITreeNodeDelete()` to delete the node in the tree.

Use the function `ProUITreeNodeExists()` to checks if the given node exists or not.

Use the function `ProUITreeNodeExpand()` to expand the node of the tree making all of its children visible.

Use the function `ProUITreeNodeHelptextGet()` to get the help text of the node in the tree.

Use the function `ProUITreeNodeIsVisible()` to checks if the node is shown or hidden.

Use the function `ProUITreeNodeIsExpanded()` to checks if the node is expanded or collapsed.

Use the function `ProUITreeNodeIsSelected()` to checks if the node is selected or not.

Use the function `ProUITreeNodeFontstyleGet()` to get the fontstyle of the node.

Use the function `ProUITreeNodeHelptextSet()` to set the helptext of the node in tree.

Use the function `ProUITreeNodeShow()` to display the node in a tree.

Use the function `ProUITreeNodeHide()` to hide the node in the tree.

Use the function `ProUITreeNodeSelect()` to select the node in the tree.

Use the function `ProUITreeNodeDeselect()` to unselect the node in the tree.

Use the function `ProUITreeNodeFontstyleSet()` to set fontstyle to the node in the tree.

Use the function `ProUITreeNodeMove()` to move the node in the tree to a child of the given parent node.

Use the function `ProUITreeNodeRename()` to rename the given node of the tree using the given name.

Use the function `ProUITreeNodetypeAdd()` to add a new node type to the tree using the information supplied in the given data structure. Use `ProUITreeNodeTypeAlloc()` to fill the `ProUITreeNodeType()`.

Use the function `ProUITreeNodetypeDelete()` to delete a node type from the tree using the information supplied in the given data structure.

Use the function `ProUITreeNodeColumntextSet()` to set the given text to be displayed in the given column corresponding to the given node of the tree.

Use the function `ProUITreeNodesSelect()` to select an array of nodes in the tree.

Use the function `ProUITreeNodetypeSelect()` to select all the nodes corresponding to a given node type.

Use the function `ProUITreeAllnodesSelect()` to select all nodes in the tree.

Use the function `ProUITreeNodesDeselect()` to unselect an array of nodes in the tree.

Use the function `ProUITreeNodetypeDeselect()` to unselect all the nodes corresponding to a given node type.

Use the function `ProUITreeAllnodesDeselect()` to unselect all the nodes in the tree.

Use the function `ProUITreeParentnameGet()` to get the name of the parent to the tree component.

Use the function `ProUITreeChildnamesGet()` to get the name of the children to the tree component.

Use the function `ProUITreeNodeVisitAction()` for visiting nodes.

Use the function `ProUITreeNodeFilterAction()` for filtering nodes.

Use the function `ProUITreeNodesVisit()` to visits all the descendent nodes of the given node in the tree.

Tree NodeType Functions

Functions Introduced

- **ProUITreeNodeTypeAlloc()**
- **ProUITreeNodeTypeFree()**
- **ProUITreeNodeTypeExpandImageSet()**
- **ProUITreeNodeTypeExpandImageGet()**
- **ProUITreeNodeTypeCollapseImageSet()**
- **ProUITreeNodeTypeCollapseImageGet()**
- **ProUITreeNodeTypePrefixSet()**
- **ProUITreeNodeTypePrefixGet()**
- **ProUITreeNodeTypeAppendStringSet()**
- **ProUITreeNodeTypeAppendStringGet()**
- **ProUITreeNodeTypeNodesSet()**
- **ProUITreeNodeTypeNodesGet()**

Use the function `ProUITreeNodeTypeAlloc()` to allocate `ProUITreeNodeType` data.

Use the function `ProUITreeNodeTypeFree()` to free `ProUITreeNodeType` data.

Use the function `ProUITreeNodeTypeExpandImageSet()` to set the image to appear when the nodetype is expanded.

Use the function `ProUITreeNodeTypeExpandImageGet()` to get the image that appears when the nodetype is expanded.

Use the function `ProUITreeNodeTypeCollapseImageSet()` to set the image to appear when the nodetype is collapsed.

Use the function `ProUITreeNodeTypeCollapseImageGet()` to get the image that appears when the nodetype is collapsed.

Use the function `ProUITreeNodeTypePrefixSet()` to set prefix to the nodetype.

Use the function `ProUITreeNodeTypePrefixGet()` to get prefix of the nodetype.

Use the function `ProUITreeNodeTypeAppendStringSet()` to append the string to the nodetype.

Use the function `ProUITreeNodeTypeAppendStringGet()` to get the appended string to the nodetype.

Use the function `ProUITreeNodeTypeNodesSet()` to set the nodes to the nodetype.

Use the function `ProUITreeNodeTypeNodesGet()` to get the nodes to the nodetype.

Tree Operations

Functions Introduced

- **ProUITreeAnchorSet()**
- **ProUITreeSizeSet()**
- **ProUITreeMinimumsizeGet()**
- **ProUITreePositionSet()**
- **ProUITreePositionGet()**
- **ProUITreeSizeGet()**
- **ProUITreeSashPositionGet()**
- **ProUITreeStateGet()**
- **ProUITreeStateSet()**
- **ProUITreeMixedStateGet()**
- **ProUITreeMixedStateSet()**
- **ProUITreeLastentereditemGet()**

Use the function `ProUITreeAnchorSet()` to set the position of the Tree with respect to a given anchor location. This function is applicable only if the parent of the Tree is a drawing area.

Use the function `ProUITreeSizeSet()` to set the size of the Tree. This field is used only if the parent is a drawing area. The function will fail, if you specify a value smaller than the minimum size for the input arguments *width* or *height*.

Use the function `ProUITreeMinimumsizeGet()` to retrieve the minimum size of the width and height of the tree in pixels. Use this function only if the parent is a drawing area.

Use the function `ProUITreePositionSet()` to set the position to the Tree with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUITreePositionGet()` to get the position to the Tree with respect to its parent. This field is used only if the parent is a drawing area.

Use the function `ProUITreeSizeGet()` to get the size of the Tree. This field is used only if the parent is a drawing area.

Use the function `ProUITreeSashPositionGet()` to get the position of the sash between the tree hierarchy and the attribute window.

Use the function `ProUITreeStateGet()` to get the state of the item in the tree. The function `ProUITreeStateSet()` sets the state of an item in the tree. The state is applicable only for a "check" type of list and refers to the checked or unchecked status of the item in the tree.

The function `ProUITreeMixedStateGet()` returns if the specified item in the tree is in an indeterminate state. Use the function `ProUITreeMixedStateSet()` to set the specified item in the tree in an indeterminate state. An indeterminate state indicates that the component is in mixed state, that is, has no selection. The state is applicable only for a "check" type of tree.

The function `ProUITreeLastentereditemGet()` returns the name of the item, which is was last pointed to by the pointer.

Tree Action Callbacks

Functions Introduced

- **ProUITreeSelectActionSet()**
- **ProUITreeActivateActionSet()**
- **ProUITreeFocusinActionSet()**
- **ProUITreeFocusoutActionSet()**
- **ProUITreeExpandActionSet()**
- **ProUITreeCollapseActionSet()**
- **ProUITreeTreecellselectActionSet()**
- **ProUITreeTreecellactivateActionSet()**
- **ProUITreeTreecellinputActionSet()**
- **ProUITreeTreecelldeleteActionSet()**
- **ProUITreeMoveActionSet()**
- **ProUITreeUpdateActionSet()**

Use the function `ProUITreeSelectActionSet()` to set the select action for a Tree. This function is called when the user changes the selected cells in the Tree.

Use the function `ProUITreeActivateActionSet()` to set the activate action for a Tree. This function is called when the user presses the return key or double-clicks the left mouse button in the Tree.

Use the function `ProUITreeFocusinActionSet()` to set the focus in action for a Tree.

Use the function `ProUITreeFocusoutActionSet()` to set the focus out action for a Tree.

Use the function `ProUITreeSelectActionSet()` to set the action function to be called when the tree is selected. The left mouse button, the `SPACE` key and the navigation keys (if the selection policy is `PROUISELPOLICY_BROWSE` or `PROUISELPOLICY_EXTENDED`) can be used to make a selection in the tree.

Use the function `ProUITreeActivateActionSet()` to set the action function to be called when the tree is activated. The tree is activated by the press of `RETURN` key or the double click of `LEFT` mouse button over the node in the tree.

Use the function `ProUITreeExpandActionSet()` to set the action function to be called when the user attempts to expand a tree node by clicking on the '+' sign.

Note

You must make this callback call `ProUITreeNodeExpand()` to actually expand the node.

Use the function `ProUITreeCollapseActionSet()` to set the action function to be called when the user attempts to collapse a tree node by clicking on the '-' sign.

Note

You must make this callback call `ProUITreeNodeCollapse()` to actually collapse the node.

Use the function `ProUITreeFocusinActionSet()` to set the action function to be called when the tree has got keyboard input focus.

Use the function `ProUITreeFocusoutActionSet()` to set the action function to be called when the tree has lost keyboard input focus.

Use the function `ProUITreeTreecellselectActionSet()` to set the action function to be called when the cell of the attribute window of the tree has been selected.

Use the function `ProUITreeTreecellactivateActionSet()` to set the action function to be called when the cell of the attribute window of the tree has been activated.

Use the function `ProUITreeTreecellinputActionSet()` to set the action function to be called when text has been entered into a cell of the attribute window of the tree.

Use the function `ProUITreeTreecelldeleteActionSet()` to set the action function to be called when the `DELETE` key has been pressed in a cell of the attribute window of the tree.

Use the function `ProUITreeMoveActionSet()` to set the action function to be called when the sash of the tree between the tree hierarchy and the attribute window has been moved.

Use the function `ProUITreeUpdateActionSet()` to set the update action for the tree.

Master Table of Resource File Attributes

Resource File Attribute	Permitted Values	Description
.ActivateOnReturn	Of Boolean type `TRUE` (default)	Flag indicating whether a `RETURN` key press should generate a `UI_ACTIVATE_ACTION` callback or whether it should cause the default button in the Dialog to be pressed.
.ArcDirection	Enumerated type `ProUIArcDirection`	The direction to use when drawing arcs in a drawing area: `PROUIARCDIR_CLOCKWISE` draws in a clockwise direction between the two line segments in order to form the arc.`PROUIARCDIR_COUNTERCLOCKWISE` Draws in a counterclockwise direction between the two line segments in order to form the arc.
.ArcFillMode	Enumerated type `ProUIArcFillMode`	The type of fill to generate for an arc drawn in a drawing area: `PROUIARCFILL_PIE`—Fills the arc bounded by two line segments joining the endpoints of the arc and its center point. `PROUIARCFILL_CHORD`—Fills the arc bounded by a single line segment that joins the two endpoints of the arc.
.Alignment	Enumerated type `ProUIAlignment`.Default is "left"	The alignment of the text:
.AttachBottom .AttachTop .AttachRight .AttachLeft	Of Boolean type `TRUE` (default), `FALSE`	The four Attach attributes specify to which sides of its grid location a component is attached. If it is attached to none the component will float inside the area available without changing size when the dialog is resized. If it is attached to the left only, for example, it will stay at the left of the area available. If it is attached both left

Resource File Attribute	Permitted Values	Description
		and right, it will stretch horizontally to fit the space available. If it is attached both top and bottom it will stretch vertically.
.Autohighlight	Of Boolean type `FALSE` (default)	Flag indicating whether to highlight the entire row when a selection is made in a cell in the row.
.BackgroundColor	Integer type based on `ProUIColorType` or a user defined color type.	The background color of the component as a user specified color or a window color of `ProUIColorType`.
.BgColor	Integer type based on `ProUIColorType` or a user defined color type.	The background drawing color of the component as a user specified color or a window color of `ProUIColorType`. This cannot be set in a resource file.
.BottomOffset .TopOffset .RightOffset .LeftOffset	Of Integer type Default Value is -1	The four Offset attributes describe the minimum gap, in pixels, between the component and the edge of the area available to it. The maximum value of an offset is 20. The default, -1, is equivalent to no offset. It is usual in Creo Parametric to put an offset of 4 around each component, so the minimum distance between components is 8 pixels. If you have a vertical column of components such as input panels in a single layout, you can reduce the overall vertical gap between them to 4 pixels.
.Bitmap	Of String type	The icon of the component. This may be a `PNG`, `PCX`, `GIF`, `BMP`, `ICO` or `CUR` file. The file must be visible to Creo Parametric by residing in Creo Parametric's text directory or in the Creo Parametric TOOLKIT application's text directory.
.ButtonStyle	Enumerated type `ProUIButtonStyle`	Following are the available types: 1. `PROUIBUTTONSTYLE_CHECK`—The horizontal and vertical margins are equal, no shadow border is drawn and by default, the button accepts keyboard input.

Resource File Attribute	Permitted Values	Description
		2. `PROUIBUTTONSTYLE_TOGGLE`—The horizontal margin is twice as wide as the vertical margin, the shadow border is always drawn and by default, the button accepts keyboard input. 3. `PROUIBUTTONSTYLE_TOOL`—The horizontal and vertical margins are equal, the shadow border is always drawn and by default, the button does not accept keyboard input. 4. `PROUIBUTTONSTYLE_FLAT`—The horizontal and vertical margins are equal, the shadow border is only drawn when the pointer is moved over the component and by default the button does not accept keyboard input. 5. `PROUIBUTTONSTYLE_LINK`—The button works like a hyperlink.
.ModalOverride	Enumerated data type `ProUIModalOverride`	ModalOverride determines whether the component is also blocked when its Dialog is blocked. It determines if there is an attempt to dismiss a blocking Dialog and menu before processing the action callbacks for the component. The following are the valid values: • `PROUIMODALOVERRIDE_NORMAL` —The component is blocked whenever the Dialog is blocked. Callbacks are processed without an attempt to dismiss a blocking Dialog and menu. • `PROUIMODALOVERRIDE_ASYNC`—The component is never blocked. Callbacks are processed without an attempt to dismiss a blocking Dialog and menu. • `PROUIMODALOVERRIDE_CAN_DISMISS_MENUS`—

Resource File Attribute	Permitted Values	Description
		The component is never blocked. Callbacks are processed only if a blocking Dialog and menu are successfully dismissed.
.CascadeDirection	Enumerated type `ProUICascadeDirection`	The direction in which the MenuPane should cascade when the CascadeButton is activated. Following are the possible values: `PROUICASCADEDIR_TOP_LEFT`—Up and to the left `PROUICASCADEDIR_TOP_RIGHT`—Up and to the right `PROUICASCADEDIR_BOTTOM_LEFT`—Down and to the left `PROUICASCADEDIR_BOTTOM_RIGHT`—Down and to the right
.ChildNames	Of String array type	Names of the children of the component. Read Only. This cannot be specified in a resource file.
.ClassName	Of String type	Name of the class of the component. It is Read-only type. These are defined in the `ProUI.h` file. This cannot be specified in a resource file.
.ClipChildren	Of Boolean type `TRUE` (default), `FALSE`	Flag indicating whether drawing operations within a DrawingArea are clipped so that they do not overlap any children contained within the DrawingArea.
.Columns	Of Integer type Default Value is 16	The width of the InputPanel, in character widths. Column widths are determines by the widest possible character in the font of the component (typically 'W'). Thus, a component with a width of 16 will likely hold words with more than 16 characters.
.ColumnLabel	Of WideString type	The tab-separated labels to be displayed as the column headers of a List.
.ColumnLabels	Of WideString type	The labels of the columns of a table.
.ColumnNames	Of String Array type	The names of the columns of a table.
.ColumnResizings	Of Integer Array type Default Value is 0	The resizing factors of the columns of a Table.

Resource File Attribute	Permitted Values	Description
.ColumnSelectionPolicy	Enumerated type `ProUISelectionpolicy`	The types of selection supported for the columns of a table: • `PROUISELPOLICY_SINGLE`—No item or one item can be selected at a time. Click to clear a selected item. • `PROUISELPOLICY_BROWSE`—Only one item can be selected at a time. You cannot clear the selection. You can replace the selected item with another selection. • `PROUISELPOLICY_MULTIPLE`—More than one item can be selected. Click to clear a selected item. • `PROUISELPOLICY_EXTENDED`—When no key is pressed, only one item can be selected at a time. You cannot clear the selection. You can replace the selected item with another selection. Press SHIFT or CTRL keys, to select multiple items. Press SHIFT key to select a range of items. Press CTRL key to select multiple items. You can click to select and clear an item.
.ColumnWidths	Of Integer Array type Default Value is the value of `DefaultColumnWidth`	The widths of the columns of a Table, in character widths. You can refer to .Columns on page
.Decorated	Boolean, default is true	Flag indicating whether the component has a decorated shadow border.
.DefaultButton	Of String type	The name of the component of the dialog which is to be treated as the default button of the dialog. The default button is automatically activated when the user hits `RETURN` or presses the middle mouse button within the Dialog.
.DefaultColumnWidth	Of Integer type Default Value is 8	The default column width if no widths are specified or if any of the ColumnWidths members are less than or equal to 0.
.Denominator	Of Integer type	The denominator value of the fractional contents of the

Resource File Attribute	Permitted Values	Description
	Default Value is 1	InputPanel.
.DialogStyle	Enumerated type `ProUIDialogStyle`	The options defined by the enumerated data type `ProUIDialogStyle` control the blocking behavior of Creo Parametric window. The dialog style types that are supported by Creo Parametric TOOLKIT application are as follows: • `PROUIDIALOGSTYLE_PARENT_MODAL`—You can use this option, when you want to block the parent window. **Note** You cannot perform the following operations after you use this option: ○ Close the newly created dialog box. ○ Click any button on the Creo user interface. ○ Close the Creo window. • `PROUIDIALOGSTYLE_WORKING`—You can use this option, if you want to block both the application and the process. The application and the process are blocked until all the pending events are processed. **Note** You can close the newly created dialog box, however, you cannot open a Creo part file with the dialog box open. • `PROUIDIALOGSTYLE_APPLICATION_MODAL`—You can use this option if you want to block the application.

Resource File Attribute	Permitted Values	Description
		Note You cannot perform the following operations after you use this option: ○ Close the newly created dialog box ○ Click any button on the Creo user interface. ○ Close the Creo window. • `PROUIDIALOGSTYLE_MODELESS`—You can use this option if you do not want to block anything. **Note** You can close the newly created dialog box, and open a Creopart file. Other dialog style values should not be used in Creo Parametric TOOLKIT dialogs. The only purpose of the other dialog style values is to be returned by the function `ProUIDialogDialogstyleGet()` on a dialog created by Creo Parametric.
.Digits	Of Integer type Default Value is -1	The number of digits to be displayed if the contents are being treated as a number. A value of 0 indicates that this attribute should be ignored when formatting the value.
.Double	Of Float type Default Value is 0	The double value of the contents of the component.
.DoubleFormat	Of String type Default Value is "%.*lf"	The format of the contents of the component if they are being treated as a double. The value of this attribute is a C formatting string, which can handle the precision and value of the contents

Resource File Attribute	Permitted Values	Description
		of the InputPanel. Note: the formatting is not applied to values entered into the component until the component's Double value is accessed programmatically.
.DoubleIncrement	Of Float type Default Value is 1	Slow increment to be used when Spinbox value is a double.
.DrawingHeight	Of Integer type Default Value is 100	Height in pixels of the drawing area.
.DrawingMode	Enumerated type `ProUIDrawingMode`	The drawing mode of the DrawingArea. Following are the possible values: `PROUIDRWMODE_COPY`—Draw using the foreground drawing color. `PROUIDRWMODE_NOT`—Draw, inverting the existing pixels of the component. `PROUIDRWMODE_AND`—Draw using a combination of the foreground drawing color AND the existing pixels. `PROUIDRWMODE_OR`—Draw using a combination of the foreground drawing color OR the existing pixels. `PROUIDRWMODE_XOR`—Draw using a combination of the foreground drawing color XOR the existing pixels.
.DrawingWidth	Of Integer type Default Value is 100	Width in pixels of the drawing area.
.Editable	Of Boolean type `TRUE` (default), `FALSE`	Flag indicating whether the text contents of the component may be modified by the user.
.FastDoubleIncrement	Of Float type Default Value is 10	The fast increment to be used when the SpinBox value is a double
.FastIncrement	Of Integer type Default Value is 10	The fast increment to be used when the SpinBox value is an integer.

Resource File Attribute	Permitted Values	Description
.FillMode	Enumerated type `ProUIFillMode`	The current drawing fill mode of the DrawingArea. Following are the possible values: `PROUIFILLMODE_SOLID`—Fill using the foreground drawing color. `PROUIFILLMODE_LIGHT_STIPPLE`— Fill using a 75% stipple of the foreground and background drawing colors. `PROUIFILLMODE_MEDIUM_STIPPLE`— Fill using a 50% stipple of the foreground and background drawing colors. `PROUIFILLMODE_HEAVY_STIPPLE`— Fill using a 25% stipple of the foreground and background drawing colors.
.Focus	Of String type	The name of the component on which the cursor is positioned when the dialog is activated. If the Dialog is already active then this specifies the name of the component within the Dialog on which on which the focus is set and the cursor is positioned.
.FontClass	Enumerated type `ProUIFontClass`	The base font class to be used to draw text in the component.
.FontSize	Of Float type Default Value is 0	Point size of the font used to draw text in the component.
.FontStyle	Enumerated type `ProUIFontStyle`	A bitwise OR of the styles of the font used to draw text in the component. Note: All styles are not supported for a given font class and platform.
.ForegroundColor	Integer type based on `ProUIColorType` or a user defined color type.	The foreground color of the component.
.FgColor	Integer type based on `ProUIColorType` or a user defined color type.	The foreground drawing color of the component as a user specified color or a window color of `ProUIColorType`.
.Height	Of Integer type Default Value is -1	Height in pixels of the component.
.HelpText	Of WideString type	The popup help-text to be

Resource File Attribute	Permitted Values	Description
		displayed whilst the pointer is over the component.
.HorzAtPoint	Of Integer type Default Value is 0	The horizontal location from the existing dialog to which a new dialog is being positioned horizontally. The possible alignment types are left/ center/ right.
.HorzDialog	Of String type	The name of the dialog to which the existing dialog is relatively being positioned horizontally.
.HorzMethod	Enumerated type `ProUIHorzPosition`	Horizontal positioning method of the dialog.
.HorzPoint	Enumerated type `ProUIHorzPosition`.	The horizontal location on thedialog which is used for positioning.
.HorzPosition	Of Integer type Default Value is -1	Desired horizontal position of the dialog. This is the absolute position.
.HorzPosOffset	Of Integer type Default Value is -1	The desired horizontal offset between the dialog to which it is being relatively positioned and the existing dialog.
.HorzSize	Of Integer type Default Value is -1	The desired width of the dialog.
.Images	Of String array typeDefault Value is `NULL`	The names of the images, which will be drawn in a DrawingArea. Each image may be a `PNG`, `PCX`, `GIF`, `BMP`, `ICO` or `CUR` file.
.Increment	Of Integer type Default Value is 1	The slow increment to be used when the SpinBox value is an integer.
.Integer	Of Integer typeDefault Value is 0	The integer value of the contents of the component.
.InputType	Enumerated type `ProUIInputtype`	Datatype of the contents of the input panel. Following are the possible types: • `PROUIINPUTTYPE_STRING` • `PROUIINPUTTYPE_WSTRING` • `PROUIINPUTTYPE_INTEGER` • `PROUIINPUTTYPE_DOUBLE` • `PROUIINPUTTYPE_FRACTION` • `PROUIINPUTTYPE_RELATION`

Resource File Attribute	Permitted Values	Description
.ItemHelpText	Of WideString Array type	The popup help-text to be displayed for the items of the component whilst the pointer is over the component. The array size should match the number of items in the component.
.ItemImage	Of String Array type	The images of the items of the component. Each image may be a `PNG`, `PCX`, `GIF`, `BMP`, `ICO` or `CUR` file. The array size should match the number of items in the component.
.Label	Of WideString type	The text shown on the component. (For dialogs: this is the dialog title).
.Labels	Of WideString type	The user visible text for each of the items of the component.
.LabelAlignment	Of Integer type Default Value is 2	The justification of the label of the Tree if the Tree has an attribute window and can be as follows: • `PROUIALIGNMENT_LEFT` • `PROUIALIGNMENT_RIGHT` • `PROUIALIGNMENT_CENTER`
.Lastentereditem	String	The name of the item, which was last pointed to by the pointer.
.Length	Of Integer type Default Value is 8	The length of the Slider, in character widths.
.ListState	Of Integer Array type	The state of each of the items of the List.
.LineStyle	Enumerated type `ProUILineStyle`	The current line drawing style of the DrawingArea. Following are the available types: • `PROUILINESTYLE_SOLID`—Draw solid lines in the foreground drawing color. • `PROUILINESTYLE_DOTTED`—Draw dotted lines in the foreground and background drawing colors. • `PROUILINESTYLE_DASHED`—Draw dashed lines in the foreground and background drawing colors.
.ListType	Enumerated type `ProUIListtype`	The possible list types are as follows:

Resource File Attribute	Permitted Values	Description
		PROUILISTTYPE_ STANDARD—No column headers or check marks.PROUILISTTYPE_ TABULATED—Display column headers, but do not display check marks. PROUILISTTYPE_CHECK— Display check marks, but do not display column headers. The value of the check mark for each item is accessed by ProUIListStateGet() and ProUIListStateSet()
.LockedColumns	Of Integer type Default Value is 0	The number of locked columns of the table.
.LockedRows	Of Integer type Default Value is 0	The number of locked rows of the Table.
.Mapped	Of Boolean type FALSE (default)	For a dialog component this flag indicates whether the dialog is visible on the screen or not. For all other components this flag indicates whether the component occupies any space when it is invisible.
.MaxDouble	Of Float type 8.507e+37	The maximum double value of the contents of the component.
.MaxInteger	Of Integer type 2147483647	The maximum integer value of the contents of the component.
.MaxLen	Of Integer type Default Value is 32	The maximum length of the text contents of the component.
.MinColumns	Of Integer type Default Value is 4	The minimum width of the component, in character widths.
.MinDouble	Of Float type -8.507e+37	The minimum double value of the contents of the component.
.MinInteger	Of Integer type -2147483647	The minimum integer value of the contents of the component.
.MinRows	Of Integer type Default Value is 1	The minimum number of visible rows of the component.
.MixedState	Of Integer Array type	The indeterminate state of the items in a Tree.
.Names	Of String Array type	The names of the items of the

Resource File Attribute	Permitted Values	Description
		component.
.Numerator	Of Integer type Default Value is 0	The numerator value of the fractional contents of the InputPanel.
.Ordinal	Of Integer type Default Value is 0	The ordinal value of the fractional contents of the InputPanel.

.Orientation	Enumerated type
	`ProUIOrientation`
		The orientation of the component. It is of the following types:
		
		`PROUI_HORIZONTAL`
		
		`PROUI_VERTICAL`

.ParentName	Of String type	The name of the parent component of the component.
.Password	Of Boolean type `FALSE` (default)	Flag indicating whether the component is used for password entry.
.PolygonFillMode	Enumerated type `ProUIPolygonFillMode`	The fill mode to be used when drawing Polygons in the DrawingArea.
.PopupMenu	Of String type	Allows you to designate the popup menu for a component This is the name of a MenuPane component to use as a popup menu for the given component.
.PopupWhenInsen	Of Boolean type	Flag indicating whether the component should display its popup menu when it is insensitive (`TRUE`) or whether no popup menu should be displayed at such times (`FALSE`).
.ProgressStyle	Enumerated type `ProUIProgressstyle`	The display style of the ProgressBar. It is of the following types: • `PROUIPROGRESS_NOTEXT` • `PROUIPROGRESS_VALUE` • `PROUIPROGRESS_PERCENT` • `PROUIPROGRESS_INTERVALS`
.RememberPosition	Of Boolean type `TRUE` (default), `FALSE`	Controls whether Creo Parametric should store the location of the dialog when it is destroyed, and apply the position to the dialog again if it is shown again.
.RememberSize	Of Boolean type `TRUE` (default), `FALSE`	Controls whether Creo Parametric should store the size the dialog when it is destroyed, and apply the size to the dialog again if it is shown.

Resource File Attribute	Permitted Values	Description
.Resizeable	Of Boolean type `TRUE` (default), `FALSE`	Identifies whether a component is allowed to resize based on changing of content. If the content is larger than the component, set this to `TRUE` to resize to contain the new content. Set it to `FALSE` to truncate or shorten the content.
.ResizableCols	Of Boolean type TRUE (default), FALSE	Flag indicating whether the Tree columns may be resized using the column headers as draggable sashes.
.Rows	Of Integer type Default Value is 8	The number of rows of the component.
.RowLabels	Of WideString Array type	The labels of the rows of the component. If labels are specified then the table will display row headers, otherwise no row headers are displayed.
.RowNames	Of String Array type	The names of the rows of the component.
.RowSelectionPolicy	Enumerated type `ProUISelectionpolicy`	The types of selection supported for the rows of a table: `PROUISELPOLICY_SINGLE`—No item or one item can be selected at a time. Click to clear a selected item. `PROUISELPOLICY_BROWSE`—Only one item can be selected at a time. You cannot clear the selection. You can replace the selected item with another selection. `PROUISELPOLICY_MULTIPLE`—More than one item can be selected. Click to clear a selected item. `PROUISELPOLICY_EXTENDED`—When no key is pressed, only one item can be selected at a time. You cannot clear the selection. You can replace the selected item with another selection. Press SHIFT or CTRL keys, to select multiple items.

Resource File Attribute	Permitted Values	Description
		Press SHIFT key to select a range of items. Press CTRL key to select multiple items. You can click to select and clear an item.
.ScrollBarsWhenNeeded	Of Boolean type	Flag indicating whether scrollbars should only be displayed when they are required (`TRUE`) or whether they should always be displayed (`FALSE`).
.SelectableNames	Of String Array type	The names of the items of the component that may be selected. An empty array indicates that every item is currently selectable.
.SelectedNames	Of String Array type	The names of the currently selected items of the component. An empty array indicates that no item of the component is selected. This attribute may not be set in the resource file.
.SelectedColumnNames	Of String Array type	The names of the currently selected columns of the Table. This attribute may not be set in the resource file.
.SelectedRowNames	Of String Array type	The names of the currently selected rows of the Table. This attribute cannot be set through the resource file.
.SelectionPolicy	Enumerated type `ProUISelectionPolicy`	The type of selection allowed for items in the component: • `PROUISELPOLICY_SINGLE`—No item or one item can be selected at a time. Click to clear a selected item. • `PROUISELPOLICY_BROWSE`—Only one item can be selected at a time. You cannot clear the selection. You can replace the selected item with another selection. • `PROUISELPOLICY_MULTIPLE`—More than one item can be selected. Click to clear a selected item. • `PROUISELPOLICY_EXTENDED`—When no key is pressed, only one item can be selected at a time. You cannot clear the selection. You can replace the selected item with another selection.

Resource File Attribute	Permitted Values	Description
		Press SHIFT or CTRL keys, to select multiple items. Press SHIFT key to select a range of items. Press CTRL key to select multiple items. You can click to select and clear an item.
.SelectByCell	Of Boolean type	Flag indicating whether the Tree attribute window allows selection by cell (`TRUE`) or whether all the cells of the selected node of the Tree are selected when a node is selected (`FALSE`).
.SelectCBRegardless	Of Boolean type	Flag indicating whether the Tree should generate a `UI_SELECT_ACTION` callback when the currently selected item is re-selected (`TRUE`) or whether such a selection should be ignored as the state has not changed.
.Sensitive	Of Boolean type	Flag indicating whether the component is disabled (`FALSE`) or sensitive to user input (`TRUE`).
.Set	Of Integer type Default Value is 0	The state of the component.
.ShowGrid	Of Boolean type	Flag indicating whether to display the grid lines of the Table.
.StartLocation	Enumerated type `ProUIStartLocation`	Following are the possible user defined start locations and anchor positions for a UI component: `PROUIDEFAULT_LOCATION`—default location `PROUITOP_LEFT`—top left-hand corner `PROUITOP_MIDDLE`—top edge `PROUITOP_RIGHT`—top right-hand corner `PROUIMIDDLE_LEFT`—left-hand edge `PROUIMIDDLE_MIDDLE`—middle `PROUIMIDDLE_RIGHT`—right-hand edge

Resource File Attribute	Permitted Values	Description
		PROUIBOTTOM_LEFT—bottom left-hand corner PROUIBOTTOM_MIDDLE—bottom edge PROUIBOTTOM_RIGHT—bottom right-hand corner
.String	Of String type	The contents of the InputPanel as a string
.TabCharsAllow	Of Boolean type	Flag indicating the behavior of the input panel when the TAB key is pressed. If the value is True, inserts a TAB key press into the text of input panel as a tab character. If the value is False, uses the TAB key press for focus traversal. The input panel ignores the tab key press. This attribute may not be set in the resource file.
.Tracking	Of Boolean type	Flag indicating whether the DrawingArea generates a UI_MOVE_ACTION whenever the pointer is moved over the visible region of the component.
.TreeAttributeWindow	Of Boolean type	Flag indicating whether the Tree has an attribute window.
.TreeBoxNodes	Of Boolean type	Flag indicating whether the nodes of the Tree should be displayed with a bounding rectangle.
.TreeCellInput	Of WideString type	The text entered by the user which caused a UI_TREE_CELL_INPUT_ACTION to be generated. This cannot be specified in the resource file.
.TreeCellSelCol	Of String type	The name of the column of the selected cell of the attribute window of the Tree. This cannot be specified in the resource file.
.TreeCellSelNode	Of String type	The name of the node of the selected cell of the attribute window of the Tree. This cannot be specified in the resource file.
.TreeChildOrder	Of Integer type Default Value is 1	The method used to order the children of a parent node of the Tree.
.TreeColumnsJusts	Of Integer Array type	The justifcations of the text of the columns of the attribute window of the Tree.
.TreeColumnsNames	Of String Array type	The names of the columns of the attribute window of the Tree.

Resource File Attribute	Permitted Values	Description
.TreeColumnsTitles	Of WideStringArray type	The labels of the column headers of the attribute window of the Tree.
.TreeColumnsWidths	Of Integer Array type	The widths of the columns of the attribute window of the Tree.
.TreeColumnOrder	Of String Array type	The display order of the columns of the attribute window of the Tree. An empty array indicates that the columns should be displayed in the order in which they were created.
.TreeCurrent	Of String type	The name of the current node of the Tree. This is the node which is drawn with a dotted focus rectangle if the Tree component has the keyboard input focus. This cannot be specified in the resource file.
.TreeDisplayRoot	Of Boolean type	Flag indicating whether the root node of the Tree should be displayed (`TRUE`) or whether it should be hidden from view (`FALSE`).
.TreeExpColNode	Of String type	The name of the node of the Tree which was expanded or collapsed to generate a `UI_EXPAND_ACTION` or a `UI_COLLAPSE_ACTION`, respectively. This cannot be specified in the resource file.
.TreeIndicateChildren	Of Boolean type	Flag indicating whether the nodes of the Tree should be displayed with a "..." suffix to indicate that they have children when the nodes themselves are not expanded.
.TreeInitialSashPercent	Of Integer type Default Value is 50	The initial percentage position of the sash of the Tree between the Tree hierarchy and the attribute window of the Tree.
.TreeKeyboardInput	Of Boolean type	Flag indicating whether the attribute window of the Tree should respond to keyboard input and generate a `UI_TREE_CELL_INPUT_ACTION`.
.TreeLinkStyle	Of Integer type Default Value is 2	The link-style used to indicate the children of a parent node of the Tree.
.TreeNodeTypeAppends	Of WideStringArray type	The text to be appended to the labels of the nodes of the node types of the Tree.
.TreeNodeTypeCollapsedImages	Of String Array type	The image to be displayed with the collapsed nodes of the node types of the Tree.

Resource File Attribute	Permitted Values	Description
.TreeNodeTypeExpandedImages	Of String Array type	The image to be displayed with the expanded nodes of the node types of the Tree.
.TreeNodeTypeHelpTexts	Of WideStringArray type	The help text of the nodes of the node types of the Tree.
.TreeNodeTypeNames	Of String Array type	The names of the node types of the Tree.
.TreeNodeTypePrefixs	Of WideStringArray type	The text to be prepended to the labels of the nodes of the node types of the Tree.
.TreeRedraw	Of Boolean type	Flag indicating whether redraws are allowed in the Tree. This cannot be specified in the resource file.
.TreeRootNode	Of String type	The name of the root node of the Tree. This cannot be specified in the resource file.
.TreeState	Of Integer Array type	The state of each of the items in the Tree.
.TreeVerticalSBPosition	Of Integer type Default Value is 3.	The position of the vertical scrollbar of the Tree.
.TruncateLabel	Of Boolean type	Flag indicating whether to truncate the labels of newly created cells in the Table to the size of their cell. Modifying this after adding rows and columns will have no effect on any existing cell's appearance.
.UnitsPerRotation	Of Integer type Default Value is 360.	The amount by which the value of the thumbwheel should be incremented for each complete revolution of the wheel.
.Value	Of WideString type	The text contents of the component.
.VertAtPoint	Enumerated type `ProUIVertPosition`.	The vertical location from the existing dialog to which a new dialog is being positioned vertically.
.VertDialog	Of String type	The name of the dialog to which the existing dialog is relatively being positioned.
.VertMethod	Enumerated type `ProUIPositioninMethod`	Vertical positioning method of the dialog.
.VertPoint	Enumerated type `ProUIVertPosition`.	The vertical location on the dialog, which is used for positioning.
.VertPosOffset	Of Integer type Default Value is -1	The desired vertical offset between the dialog to which it is being relatively positioned and the existing dialog.
.VertSize	Of Integer type	The desired height of the dialog.

Resource File Attribute	Permitted Values	Description
	Default Value is -1	
.Visible	Of Boolean type	Flag indicating whether the component is shown (`TRUE`) or hidden (`FALSE`).
.VisibleNames	Of String Array type	The names of the visible items of the component. An empty array indicates that every item of the component is visible.
.VisibleRows	Of Integer type Default Value is 4	The number of visible rows of the component.
.WideString	Of WideString type	The contents of the InputPanel as a wide-string.
.Width	Of Integer type Default Value is -1	Width of the component, in pixels.

Using Resource Files

Resource files are text files that describe the overall structure of a dialog box.

Note

For most Creo Parametric TOOLKIT applications, it is not required to use resource files. Instead you can use the functions described in the previous sections to create, lay out, and populate dialog boxes directly. Information in this section is provided for the few applications that still require resource files.

When the Creo Parametric TOOLKIT application wants to show a dialog box to the Creo Parametric user, it simply asks Creo Parametric to load the dialog box from the resource file. The first task for the Creo Parametric TOOLKIT user who wants to display a dialog box is to write the resource file.

The resource file describes:

- Overall attributes of the dialog box.
- A list of components it contains.
- Attributes of the components themselves and the relative positions of the components.
- Rules for how they behave when the user resizes the dialog box.

Many of the dialog box and component attributes can also be read and modified programmatically with Creo Parametric TOOLKIT functions.

A resource file must be called `dialog_name.res` where `dialog_name` is the name of the dialog box. The name of the resource file should be in the lower case. Resource files used by the Creo Parametric TOOLKIT application must be stored under the `text` directory referred to by the `text_dir` statement in the Creo Parametric TOOLKIT registry file.

Location and Translation of Resource Files

If the application uses language-specific directories, the resource file must be located in the `<application_text_dir>/<language_dir>/resource` directory. One resource file must exist for each language supported by the application. If the application supports only one language, then the resource file may be located in the `<application_text_dir>/resource` directory.

A Creo Parametric TOOLKIT application can optionally include only one resource file for all supported languages in the `<application_text_dir>/resource` directory. This application should also include a text file containing the translated entities and translated text for each additional language, in the format used by the function `ProMessageDisplay()`, in the `<application_text_dir>/resource/<language_dir>` directory. The translation text file should include the following items:

- The component and attribute for each translated item in line 1
- The English text in line 2
- The translated entities and translated text in line 3
- Line 4 should be left blank

Function Introduced:

- **ProUITranslationFilesEnable()**

Use the function `ProUITranslationFilesEnable()` to set your Creo Parametric TOOLKIT application to use the single resource file and multiple translation files method of deploying resource files for translated dialog boxes. Call this function from your application's `user_intialize()` function before any call is made to another `ProUI*` function.

Syntax of Resource Files

The resource file is composed of nested statements; each statement is enclosed in parentheses, and contains either a keyword or the name of a dialog or component attribute followed by one or more values and/or other statements.

The top-level statement in a resource file for dialog must always be

```
(
Dialog dialog_name     (other statements...   )
)
```

where `dialog_name` is the name of the dialog itself. The dialog name is used to refer to the dialog from the source code of the application. The name of a dialog can be of any length, and contains alphanumeric characters and underscores. Case is ignored.

If the resource file contains only a layout, the top-level statement would be

```
(
Layout layout_name     (other statements...   )
)
```

instead. Collectively, the top statement describes the outermost container in the resource file.

The two statements that follow the name of the container are always Components and Resources.

The Components statement simply lists the types and names of the components that the dialog contains.

Components of type Layout, MenuPane, CascadeButton, Table, or Tree may have their own container-level statements following the Dialog statement. The formats of these statements are exactly the same as the Dialog statement.

If one of the components is a tab, the layouts that the tab contains are listed after the tab name in the Components part of the Dialog statement. Similarly, if one of the components is a MenuBar or a CascadeButton, the MenuPanes that it contains are listed after the MenuBar name in the Components part of the Dialog statement.

A Resources statement contains one or more attribute assignments. Each attribute assignment statement is of this form

```
(componentname.attributename    value)
```

The Resource statement defines the attributes of the dialog and the components, and the Layout which defines the relative positions of the components, and the way in which they behave when the dialog is resized. If the component name is missing, the statement (for example, (`.attributename value`)) will be assumed to apply to an attribute of the dialog or layout itself. The attributes and their values are described in detail in the following sections of this chapter.

The last attribute statement in a Resources statement is a special one called `.Layout`. The value of a `.Layout` statement is always a single statement of type Grid. The Grid statement describes a flexible grid of rows and columns in which the components are placed; this defines the neighbor-relations between components, in other words their relative positions. The absolute positions, and the sizes, of the components may change as the grid stretches and shrinks in response to the user resizing the window containing the dialog. Components of type MenuPane do not require a `.Layout` statement.

The Grid statement contains the following values:

- Rows statement—lists the rows in the grid.
- Cols statement—lists the columns in the grid.
- Values list—specify the contents of each grid location in turn, reading left-to-right, top-to-bottom. Each of the values can be either
  - The name of a component
  - A Pos statement, to specify the row and column number of the next component in the list, if it is not in the next available location. A Pos statement allows you to skip some locations, leaving them empty.
  - Another Grid statement, to show that the location contains several components on their own local grid.

The value of a Rows or Cols statement is a list of integers, one for each row or column. The value of the integer is 0 if the row or column cannot be resized when user resized the dialog. the integer value is 1 if it can be resized. (It is normal to set this to 1, so that all the components in the dialog will stretch when the dialog stretches.)

The size and position of the components within the grid is also partly determined by the values set for the `Attach` and `Offset` attributes described in more detail in the section Master Table of Resource File Attributes on page 425.

Note

Ensure that the resource file contains a new line after the last parenthesis, or else the function `ProUIDialogCreate()` will return an error.

Example 8: Dialog with All Components

```
(Dialog allcomponents
    (Components
        (PushButton                     PushButton1)
        (PushButton                     PushButton2)
        (Tab                            Tab3
                                        Layout11
                                        Layout8)
        (Label                          Label16)
    )
    (Resources
        (PushButton1.Label              "OK")
        (PushButton1.TopOffset          4)
        (PushButton1.BottomOffset       4)
        (PushButton1.LeftOffset         4)
        (PushButton1.RightOffset        4)
        (PushButton2.Label              "Cancel")
        (PushButton2.TopOffset          4)
```

```
        (PushButton2.BottomOffset       4)
        (PushButton2.LeftOffset         4)
        (PushButton2.RightOffset        4)
        (Tab3.Decorated                 True)
        (Label16.Label                  "Push Buttons")
        (.Label                         "All Components")
        (.Layout
            (Grid (Rows 1 1 1) (Cols 1)
                Tab3
                Label16
                (Grid (Rows 1) (Cols 1 1)
                    PushButton1
                    PushButton2
                )
            )
        )
    )
)
(Layout Layout11
    (Components
        (SubLayout                      Layout1)
    )

    (Resources
        (.Label           "Tab with two layouts")
        (.Decorated                     True)
        (.Layout
            (Grid (Rows 1) (Cols 1)
                Layout1
            )
        )
    )
)
(Layout Layout1
    (Components
        (InputPanel                     InputPanel1)
        (Label                          Label1)
        (CheckButton                    CheckButton1)
        (Label                          Label2)
        (Label                          Label3)
        (Label                          Label4)
        (List                           List1)
        (Label                          Label5)
        (OptionMenu                     OptionMenu1)
        (Label                          Label6)
        (Label                          Label8)
        (RadioGroup                     RadioGroup1)
        (Slider                         Slider1)
        (Label                          Label9)
        (TextArea                       TextArea1)
        (Label                          Label10)
```

```
    (CheckButton                    CheckButton3)
    (Label                          Label7)
    (Label                          Label11)
    (ThumbWheel                     ThumbWheel1)
    (ProgressBar                    ProgressBar1)
    (SpinBox                        SpinBox1)
    (Label                          Label14)
    (Separator                      Separator1)
    (Label                          Label13)
    (Table                          Table1)
    (Label                          Label17)
)
(Resources
    (InputPanel1.InputType          2)
    (Label1.Label                   "Input Panel")
    (Label1.AttachLeft              True)
    (CheckButton1.Label             "Check2")
    (CheckButton1.Set               True)
    (CheckButton1.TopOffset         4)
    (CheckButton1.BottomOffset      4)
    (CheckButton1.LeftOffset        4)
    (CheckButton1.RightOffset       4)
    (Label2.Label                   "Check Buttons")
    (Label2.AttachLeft              True)
    (Label2.TopOffset               4)
    (Label2.BottomOffset            4)
    (Label2.LeftOffset              4)
    (Label2.RightOffset             4)
    (Label3.Label                   "Label")
    (Label3.AttachLeft              True)
    (Label4.Label                   "Label Text")
    (List1.Names                    "n1"
                                    "n2"
                                    "n3"
                                    "n4"
                                    "n5")
    (List1.Labels                   "Value 1"
                                    "Value 2"
                                    "Value 3"
                                    "Value 4"
                                    "Value 5")
    (Label5.Label                   "List")
    (Label5.AttachLeft              True)
    (OptionMenu1.Names              "n1"
                                    "n2"
                                    "n3"
                                    "n4"
                                    "n5")
    (OptionMenu1.Labels             "Option 1"
                                    "Option 2"
                                    "Option 3"
```

```
                                "Option 4"
                                "Option 5")
(Label6.Label                   "Option Menu")
(Label6.AttachLeft              True)
(Label8.Label                   "Radio Group")
(Label8.AttachLeft              True)
(RadioGroup1.Orientation        True)
(RadioGroup1.AttachLeft         False)
(RadioGroup1.AttachRight        False)
(RadioGroup1.AttachTop          False)
(RadioGroup1.AttachBottom       False)
(RadioGroup1.Names              "n1"
                                "n2"
                                "n3")
(RadioGroup1.Labels             "Rad1"
                                "Rad2"
                                "Rad3")
(RadioGroup1.Alignment          2)
(Label9.Label                   "Slider")
(Label9.AttachLeft              True)
(Label10.Label                  "Text Area")
(Label10.AttachLeft             True)
(CheckButton3.Label             "Check1")
(Label7.Label                   "Thumbwheel")
(Label7.AttachLeft              True)
(Label11.Label                  "Progress bar")
(Label11.AttachLeft             True)
(Label14.Label                  "Spinbox")
(Label14.AttachLeft             True)
(Label13.Label                  "Separator")
(Label13.AttachLeft             True)
(Table1.RowNames                "r1"
                                "r2"
                                "r3")
(Table1.RowLabels               "1"
                                "2"
                                "3")
(Table1.ColumnNames             "c1"
                                "c2"
                                "c3")
(Table1.ColumnLabels            "- A -"
                                "- B -"
                                "- C -")
(Table1.ColumnWidths            4
                                4
                                4)
(Table1.VisibleRows             4)
(Table1.ShowGrid                True)
(Label17.Label                  "Table")
(Label17.AttachLeft             True)
(.Label                         "Decorated layout")
```

```
        (.Decorated                     True)
        (.TopOffset                     4)
        (.BottomOffset                  4)
        (.LeftOffset                    4)
        (.RightOffset                   4)
        (.Layout
            (Grid (Rows 1 1 1 1 1 1 1 1 1 1 1 1) (Cols 1 1)
                Label2
                (Grid (Rows 1 1) (Cols 1)
                    CheckButton3
                    CheckButton1
                )
                Label1
                InputPanel1
                Label3
                Label4
                Label5
                List1
                Label6
                OptionMenu1
                Label11
                ProgressBar1
                Label8
                RadioGroup1
                Label13
                Separator1
                Label9
                Slider1
                Label14
                SpinBox1
                Label17
                Table1
                Label10
                TextArea1
                Label7
                ThumbWheel1
            )
        )
    )
)

(Layout Layout8
   (Components
      (Label                          Label15)
    )
    (Resources
        (Label15.Bitmap         "ptc_logo")
        (.Label                 "Second layout")
        (.Layout
            (Grid (Rows 1) (Cols 1)
                Label15
```

```
            )
        )
    )
)
```

Example 9: Resource File for Dialog with Four Components on 2x2 Grid

This example shows a simple dialog which contains four components on a single 2-by-2 grid.

```
(Dialog Simple
     (Components
         (PushButton                    OK)
         (PushButton                    Cancel)
         (Label                         RadioLabel)
         (RadioGroup                    Tolgroup)
     )
    (Resources
         (OK.Label                      "OK")
         (OK.TopOffset                  4)
         (OK.BottomOffset               4)
         (OK.LeftOffset                 4)
         (OK.RightOffset                4)
         (Cancel.Label                  "Cancel")
         (Cancel.TopOffset              4)
         (Cancel.BottomOffset           4)
         (Cancel.LeftOffset             4)
         (Cancel.RightOffset            4)
         (RadioLabel.Label              "Value to use")
         (RadioLabel.TopOffset          4)
         (RadioLabel.BottomOffset       4)
         (RadioLabel.LeftOffset         4)
         (RadioLabel.RightOffset        4)
         (Tolgroup.Orientation          True)
         (Tolgroup.Names                "Top"
                                        "Middle"
                                        "Bottom")
         (Tolgroup.Labels               "Upper limit"
                                        "Nominal"
                                        "Lower limit")
         (.Label                        "Simple Dialog")
         (.Layout
             (Grid (Rows 1 1) (Cols 1 1)
                 RadioLabel
                 Tolgroup
                 OK
                 Cancel
             )
         )
     )
```

```
)
```

Example 10: Resource File for Subgrid Dialog

This example shows the resource file for the preceding subgrid dialog.

```
(Dialog subgrid
     (Components
         (PushButton                    OK)
         (PushButton                    Cancel)
         (InputPanel                    FeatNamePanel)
         (Label                         FeatNameLabel)
     )
    (Resources
         (OK.Label                      "OK")
         (OK.TopOffset                  4)
         (OK.BottomOffset               4)
         (OK.LeftOffset                 4)
         (OK.RightOffset                4)
         (Cancel.Label                  "Cancel")
         (Cancel.TopOffset              4)
         (Cancel.BottomOffset           4)
         (Cancel.LeftOffset             4)
         (Cancel.RightOffset            4)
         (FeatNamePanel.TopOffset       4)
         (FeatNamePanel.BottomOffset    4)
         (FeatNamePanel.LeftOffset      4)
         (FeatNamePanel.RightOffset     4)
         (FeatNameLabel.Label           "Feature name")
         (.Label                        "Subgrid")
         (.Layout
             (Grid (Rows 1 1 1) (Cols 1)
                 FeatNameLabel
                 FeatNamePanel
                 (Grid (Rows 1) (Cols 1 1)
                     OK
                     Cancel
                 )
             )
         )
     )
)
```

Example 11: Resource File for Subgrid Dialog with Resize

A better way to lay out the dialog in the previous example would be to place the input panel text to the left of the input panel. If this were done with a single 2-by-2 grid, the input panel label and panel itself would resize to take up the same width

as the **OK** and **Cancel** buttons. To avoid this you can put the first two components in a layout of their own, which therefore has its own grid, so the columns can take up their own widths.

This example demonstrates this method.

```
(Dialog subgrid
     (Components
         (PushButton                        OK)
         (PushButton                        Cancel)
         (InputPanel                        FeatNamePanel)
         (Label                             FeatNameLabel)
     )
    (Resources
         (OK.Label                          "OK")
         (OK.TopOffset                      4)
         (OK.BottomOffset                   4)
         (OK.LeftOffset                     4)
         (OK.RightOffset                    4)
         (Cancel.Label                      "Cancel")
         (Cancel.TopOffset                  4)
         (Cancel.BottomOffset               4)
         (Cancel.LeftOffset                 4)
         (Cancel.RightOffset                4)
         (FeatNamePanel.TopOffset           4)
         (FeatNamePanel.BottomOffset        4)
         (FeatNamePanel.LeftOffset          4)
         (FeatNamePanel.RightOffset         4)
         (FeatNameLabel.Label               "Feature name")
         (.Label                            "Subgrid")
         (.Layout
             (Grid (Rows 1 1 1) (Cols 1)
                 FeatNameLabel
                 FeatNamePanel
                 (Grid (Rows 1) (Cols 1 1)
                     OK
                     Cancel
                 )
             )
         )
     )
 )
```

Example 12: Resource File with Offsets, Attachments, and Help Text

This example shows a section of resource file which specifies offsets, attachments, and help text is shown below. This is an extended version of the previous example.

```
    (Resources
         (FeatNameLabel.Label               "Feature name")
```

```
        (FeatNameLabel.AttachLeft       True)
        (FeatNameLabel.TopOffset        4)
        (FeatNameLabel.BottomOffset     4)
        (FeatNameLabel.LeftOffset       4)
        (FeatNameLabel.RightOffset      4)
        (FeatNamePanel.HelpText         "Enter the Feature name here.")
        (FeatNamePanel.TopOffset        4)
        (FeatNamePanel.BottomOffset     4)
        (FeatNamePanel.LeftOffset       4)
        (FeatNamePanel.RightOffset      4)
        ....
    )
```

The .Bitmap attribute is the name of a file which contains a bitmap description of an image which should be applied to the component. The file can be one of the following types:

- .GIF
- .PNG
- .PCX
- .BMP
- .ICO
- .CUR

If you supply only the root of the filename, the .bif format will be assumed.

Example 13: Resource File with Text Question, OK and Cancel Buttons

This example shows the use of a simple dialog, which contains a text question and **OK** and **Cancel** buttons. It would be used for any kind of confirmation request.

For a programatic method of creating this dialog box refer to Example 1: Source for Dialog with Text Question, OK and Cancel Buttons on page 355.

```
(Dialog confirm
     (Components
         (Label                          Question)
         (PushButton                     OK)
         (PushButton                     Cancel)
     )
    (Resources
         (Question.Label                 "Dummy label")
         (Question.TopOffset             4)
         (Question.BottomOffset          4)
         (Question.LeftOffset            4)
         (Question.RightOffset           4)
         (OK.Label                       "OK")
         (OK.TopOffset                   4)
         (OK.BottomOffset                4)
```

```
        (OK.LeftOffset                  4)
        (OK.RightOffset                 4)
        (Cancel.Label                   "Cancel")
        (Cancel.TopOffset               4)
        (Cancel.BottomOffset            4)
        (Cancel.LeftOffset              4)
        (Cancel.RightOffset             4)
        (.Label                         "Confirm")
        (.Layout
            (Grid (Rows 1 1) (Cols 1)
                Question
                (Grid (Rows 1) (Cols 1 1)
                    OK                      Cancel
                )
            )
        )
    )
)
```

Example 14: UI List Resource File

The example below shows the resource file used to create UI List.For a programatic method of creating the UI List refer to Example 2: To use UI List Functions on page 381.

```
(Dialog ug_uilist
    (Components
        (SubLayout                      ug_uilist_layout1)
        (SubLayout                      ug_uilist_layout2)
    )

    (Resources
        (.Label                         "UI List")
        (.Layout
            (Grid (Rows 1 1) (Cols 1)
                ug_uilist_layout1
                ug_uilist_layout2
            )
        )
    )
)

(Layout ug_uilist_layout1
    (Components
        (List                           ug_uilist_comp)
        (Label                          ug_uilist_label_1)
        (TextArea                       ug_uilist_txtarea)
        (Label                          ug_uilist_lable_2)
    )

    (Resources
```

```
        (ug_uilist_label_1.Label        "List area")
        (ug_uilist_txtarea.FontStyle    4)
        (ug_uilist_lable_2.Label        "Text Area")
        (.Decorated                     True)
        (.DecoratedStyle                3)
        (.Layout
            (Grid (Rows 1 1 1 1) (Cols 1)
                ug_uilist_label_1
                ug_uilist_comp
                ug_uilist_lable_2
                ug_uilist_txtarea
            )
        )
    )
)

(Layout ug_uilist_layout2
    (Components
        (PushButton                     ug_uilist_ok)
        (PushButton                     ug_uilist_cancel)
    )

    (Resources
        (ug_uilist_ok.Label             "Ok")
        (ug_uilist_ok.TopOffset         4)
        (ug_uilist_ok.BottomOffset      4)
        (ug_uilist_ok.RightOffset       20)
        (ug_uilist_cancel.Label         "Cancel")
        (ug_uilist_cancel.TopOffset     4)
        (ug_uilist_cancel.BottomOffset  4)
        (ug_uilist_cancel.LeftOffset    20)
        (.Layout
            (Grid (Rows 1) (Cols 1 1 1)
                ug_uilist_ok
                (Pos 1 3)
                ug_uilist_cancel
            )
        )
    )
)
```

Example 15: Resource File for Dialog with Menubar

The example below shows the resource file used to create the dialog with MenuBars as described in section Example 9: Resource File for Dialog with Four Components on 2x2 Grid on page 452.

```
(Dialog menus
    (Components
        (MenuBar                        MenuBar1
                                        MenuPane1
```

```
                                        MenuPane2)
        (TextArea                       TextArea1)
    )

    (Resources
        (TextArea1.Rows                 10)
        (TextArea1.Columns              25)
        (.Label                         "Menubar dialog")
        (.Layout
            (Grid (Rows 1) (Cols 1)
                TextArea1
            )
        )
    )
)

(MenuPane MenuPane1
    (Components
        (PushButton                     PushButton1)
        (PushButton                     PushButton2)
    )

    (Resources
        (.Label                         "MenuPane1")
  (PushButton1.Label              "Button1")
  (PushButton2.Label              "Button2")
    )
)

(MenuPane MenuPane2
    (Components
        (PushButton                     PushButton3)
        (Separator                      Separator1)
        (RadioGroup                     RadioGroup1)
        (RadioGroup                     RadioGroup2)
        (Separator                      Separator2)
        (CascadeButton                  CascadeButton2
                                        MenuPane4)
        (CheckButton                    CheckButton1)
    )

    (Resources
  (.Label                         "MenuPane2")
        (PushButton3.Label              "Pushbutton")
        (RadioGroup1.Names              "RG1"
                                        "RG2")
        (RadioGroup1.Labels             "Radio group 1"
                                        "Radio group 2")
        (CascadeButton2.Label           "Cascade button")
        (CheckButton1.Label             "Checkbutton")
    )
```

```
)

(MenuPane MenuPane4
    (Components
        (PushButton                     PushButton5)
        (PushButton                     PushButton6)
    )

    (Resources
        (PushButton5.Label              "Pushbutton")
        (PushButton6.Label              "Pushbutton")
    )
)
```

Example 16: Progress Bar Resource File

```
(Dialog progressbars
    (Components
        (ProgressBar                    ProgressBar2)
        (ProgressBar                    ProgressBar3)
        (ProgressBar                    ProgressBar4)
        (ProgressBar                    ProgressBar5)
        (Label                          Label2)
        (Label                          Label3)
        (Label                          Label4)
        (Label                          Label5)
        (Separator                      Separator1)
        (SpinBox                        SpinBox1)
        (SpinBox                        SpinBox2)
        (ThumbWheel                     ThumbWheel1)
    )

    (Resources
        (ProgressBar2.ProgressStyle     0)
        (ProgressBar2.TopOffset         4)
        (ProgressBar2.BottomOffset      4)
        (ProgressBar2.LeftOffset        4)
        (ProgressBar2.RightOffset       4)
        (ProgressBar3.ProgressStyle     1)
        (ProgressBar3.TopOffset         4)
        (ProgressBar3.BottomOffset      4)
        (ProgressBar3.LeftOffset        4)
        (ProgressBar3.RightOffset       4)
        (ProgressBar4.TopOffset         4)
        (ProgressBar4.BottomOffset      4)
        (ProgressBar4.LeftOffset        4)
        (ProgressBar4.RightOffset       4)
        (ProgressBar5.ProgressStyle     3)
        (ProgressBar5.TopOffset         4)
        (ProgressBar5.BottomOffset      4)
        (ProgressBar5.LeftOffset        4)
```

```
        (ProgressBar5.RightOffset       4)
        (Label2.Label                   "No text")
        (Label2.AttachLeft              True)
        (Label3.Label                   "Value")
        (Label3.AttachLeft              True)
        (Label4.Label                   "Percentage")
        (Label4.AttachLeft              True)
        (Label5.Label                   "Interval")
        (Label5.AttachLeft              True)
        (Separator1.Orientation         True)
        (SpinBox1.Columns               8)
        (SpinBox1.Editable              False)
        (SpinBox1.IntegerBase           8)
        (SpinBox1.Rate                  1)
        (SpinBox1.InputType             3)
        (SpinBox1.Digits                3)
        (SpinBox1.DoubleFormat          "%12.*le")
        (ThumbWheel1.DecoratedStyle     4)
        (ThumbWheel1.InfiniteRange      False)
        (.Layout
            (Grid (Rows 1 1 1 1 1 1) (Cols 1 1 1)
                (Pos 1 2)
                Label2
                ProgressBar2
                (Pos 2 2)
                Label3
                ProgressBar3
                (Pos 3 2)
                Label4
                ProgressBar4
                (Pos 5 1)
                Label5
                Separator1
                ProgressBar5
                SpinBox1
                SpinBox2
                ThumbWheel1
            )
        )
    )
)
```

Example 17: Component Visibility Resource File

For a programatic method of creating this dialog box refer to Example 3: Controlling Component Visibility or Sensitivity at Runtime on page 390.

```
(Dialog uguivisibility
    (Components
      (SubLayout                        PushbuttonVisibility)
```

```
      (PushButton                          CloseButton)
)
    (Resources
      (CloseButton.Label                    "Close")
      (CloseButton.TopOffset                4)
      (CloseButton.BottomOffset             4)
      (CloseButton.LeftOffset               4)
      (CloseButton.RightOffset              4)
      (.Label                               "Component Visibility")
      (.Layout
          (Grid (Rows 1 1) (Cols 1)
                                             PushbuttonVisibility
CloseButton
              )
         )
     )
 )

    (Layout                                   PushbuttonVisibility
      (Components
         (PushButton                         TargetBtn)
         (CheckButton                        VisibleCheck)
         (CheckButton                        SensitiveCheck)
         (InputPanel                         ButtonLabel)
     )
    (Resources
         (TargetBtn.Label                    "PushButton")
         (TargetBtn.TopOffset                4)
         (TargetBtn.BottomOffset             4)
         (TargetBtn.LeftOffset               4)
         (TargetBtn.RightOffset              4)
         (TargetBtn.Resizeable               True)
         (VisibleCheck.Label                 "Visible")
         (VisibleCheck.Set                    True)
         (VisibleCheck.TopOffset              4)
         (VisibleCheck.BottomOffset           4)
         (VisibleCheck.LeftOffset             4)
         (VisibleCheck.RightOffset            4)
         (SensitiveCheck.Label                "Sensitive")
         (SensitiveCheck.Set                  True)
         (SensitiveCheck.TopOffset            4)
         (SensitiveCheck.BottomOffset         4)
         (SensitiveCheck.LeftOffset           4)
         (SensitiveCheck.RightOffset          4)
         (ButtonLabel.Value                   "PushButton")
         (ButtonLabel.TopOffset               4)
         (ButtonLabel.BottomOffset            4)
         (ButtonLabel.LeftOffset              4)
         (ButtonLabel.RightOffset             4)
         (.Label                           "PushButtons")
         (.Layout
```

```
            (Grid (Rows 1) (Cols 1 1 1 1)
                TargetBtn
                VisibleCheck
                SensitiveCheck
                ButtonLabel
            )
        )
    )
)
```

Example 18: Resource File for Dialog with Slider and Linked InputPanel

This example shows the resource file for a dialog with slider and linked input panel. For a programatic method of creating this dialog box refer to Example 4: Source of Dialog with Slider and Linked InputPanel on page 395.

```
(Dialog angle
     (Components
         (SubLayout                      Layout1)
         (SubLayout                      Layout2)
     )
    (Resources
         (.Label                         "Angle")
         (.DefaultButton                 "OK")
         (.Layout
             (Grid (Rows 1 1) (Cols 1)
                 Layout1                 Layout2
             )
         )
     )
)
(Layout Layout1
     (Components
         (Slider                         Slider)
         (InputPanel                     InputPanel)
         (Label                          Prompt)
     )
    (Resources
         (Slider.MinInteger              -180)
         (Slider.MaxInteger              180)
         (Slider.Length                  12)
         (Slider.Tracking                True)
         (Slider.TopOffset               4)
         (Slider.BottomOffset            4)
         (Slider.LeftOffset              4)
         (Slider.RightOffset             4)
         (InputPanel.Columns             4)
         (InputPanel.AttachLeft          False)
         (InputPanel.AttachRight         False)
         (InputPanel.MinInteger          -180)
```

```
        (InputPanel.MaxInteger          180)
        (InputPanel.TopOffset           4)
        (InputPanel.BottomOffset        4)
        (InputPanel.LeftOffset          4)
        (InputPanel.RightOffset         4)
        (InputPanel.InputType           2)
        (Prompt.Label                   "Dummy text")
        (.Layout
            (Grid (Rows 1 1) (Cols 1)
                Prompt
               (Grid (Rows 1) (Cols 1 1)
                    Slider
                    InputPanel
               )
            )
        )
    )
)
(Layout Layout2
    (Components
        (PushButton                     OK)
        (PushButton                     Cancel)
    )
    (Resources
        (OK.Label                       "OK")
        (OK.TopOffset                   4)
        (OK.BottomOffset                4)
        (OK.LeftOffset                  4)
        (OK.RightOffset                 4)
        (Cancel.Label                   "Cancel")
        (Cancel.TopOffset               4)
        (Cancel.BottomOffset            4)
        (Cancel.LeftOffset              4)
        (Cancel.RightOffset             4)
        (.Layout
            (Grid (Rows 1) (Cols 1 1)
                OK                Cancel
            )
        )
    )
)
```

Example 19: UG Tables Resource File

The following example shows a resource file containing a table. Components may be assigned to a table through the table layout. Components of the table layout are invisible until assigned or copied in to a table cell.

```
(Dialog uguitableexample
    (Components
        (SubLayout                      TableLayout)
```

```
        (SubLayout                          ButtonLayout)
        (OptionMenu                         ToCopy)
    )

    (Resources
        (ToCopy.Visible                     False)
        (ToCopy.AttachTop                   True)
        (ToCopy.AttachBottom                True)
        (ToCopy.Names                       "ONE"
                                            "MANY")
        (ToCopy.Labels                      "Select One"
                                            "Select Many")
        (.Label                             "UG Tables")
        (.Layout
            (Grid (Rows 1 0 1) (Cols 1)
                TableLayout
                ButtonLayout
                ToCopy
            )
        )
    )
)

(Layout TableLayout
    (Components
        (Table                              LargeTable)
    )

    (Resources
        (LargeTable.Columns                 40)
        (LargeTable.MinRows                 4)
        (LargeTable.TopOffset               4)
        (LargeTable.BottomOffset            4)
        (LargeTable.LeftOffset              4)
        (LargeTable.RightOffset             4)
        (LargeTable.RowNames                "A"
                                            "B"
                                            "C"
                                            "D"
                                            "E")
        (LargeTable.ColumnNames             "1"
                                            "2"
                                            "3"
                                            "4")
        (LargeTable.RowLabels               "Alpha"
                                            "Beta"
                                            "Gamma"
                                            "Delta"
                                            "Epsilon")
        (LargeTable.ColumnLabels            "One"
                                            "Two"
```

```
                                        "Three"
                                        "Four")
        (LargeTable.ShowGrid            True)
        (.AttachLeft                    True)
        (.AttachRight                   True)
        (.AttachTop                     True)
        (.AttachBottom                  True)
        (.Layout
            (Grid (Rows 1) (Cols 1)
                LargeTable
            )
        )
    )
)

(TableLayout LargeTable
    (Components
        (PushButton                     BaseButton)
        (CheckButton                    BaseCheckButton)
        (InputPanel                     BaseInputPanel)
    )

    (Resources
        (BaseButton.Label               "Table Button")
        (BaseCheckButton.Label          "Table Check Button")
        (BaseInputPanel.Value           "Table Input Panel")
    )
)

(Layout ButtonLayout
    (Components
        (PushButton                     CloseButton)
        (PushButton                     ModifySelectButton)
        (Label                          Label1)
    )

    (Resources
        (CloseButton.Label              "Close")
        (CloseButton.TopOffset          4)
        (CloseButton.BottomOffset       4)
        (CloseButton.LeftOffset         4)
        (CloseButton.RightOffset        4)
        (ModifySelectButton.Label       "Toggle cells")
        (ModifySelectButton.Visible     False)
        (ModifySelectButton.TopOffset   4)
        (ModifySelectButton.BottomOffset 4)
        (ModifySelectButton.LeftOffset  4)
        (ModifySelectButton.RightOffset 4)
        (.Layout
            (Grid (Rows 1 1) (Cols 1 1 1)
                (Pos 1 3)
```

```
                Label1
                ModifySelectButton
                CloseButton
            )
        )
    )
)
```

Example 20: UG Tables Component Resource File

The following example demonstrates different methods that can be used to assign components into table cells. The example directly assigns a stored component, copies a stored component, copies a component from elsewhere in the dialog, and copies a component from another independent dialog.

The Example 19: UG Tables Resource File on page 463 is the primary resource file for the following example. The table components dialog is another dialog from which the following resource code will copy a component into the table.

For a programatic method of assigning components into table cells refer to Example 5: To Assign Components into Table Cells on page 404.

```
(Dialog uguitablecomponents
    (Components
        (PushButton                     ExternalButtonToCopy)
    )

    (Resources
        (ExternalButtonToCopy.Label     "From other dialog...")
        (.Layout
            (Grid (Rows 1) (Cols 1)
                ExternalButtonToCopy
            )
        )
    )
)
```

19

User Interface: Dashboards

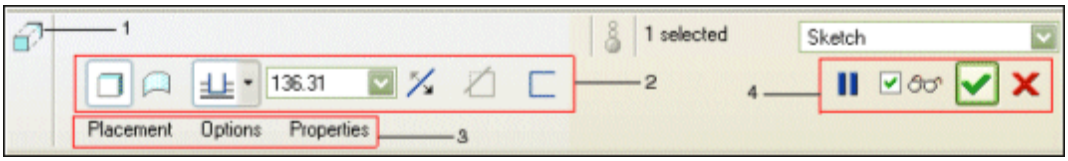
This chapter introduces the basics of dashboards and describes how Creo Parametric TOOLKIT offers the capability to create and control dashboards. Creo Parametric TOOLKIT applications can construct dashboards for any tool where they would typically require a dialog box.

The behavior of the Creo Parametric TOOLKIT dashboard is similar to the Creo Parametric dashboard. The TOOLKIT dashboards use the same commands and dismiss mechanism as the native Creo Parametric dashboard. However, individual commands may have different accessibility logic and so different dashboards can have slightly different commands available.

Introduction to Dashboards

A dashboard is an embedded "dialog box" at the top of the Creo Parametric graphics window. A dashboard typically appears when you create or modify a feature in Creo Parametric. It offers the necessary controls, inputs, status and guidance for creating or editing of features.

Dashboard components



1 Tool Icon
2 Dialog Bar
3 Slide-down Panels
4 Standard Buttons

A dashboard consists of the following components:

- A main dialog bar, which show the commonly used commands and entry fields. You perform most of your modeling tasks in the graphics window and the dialog bar. When you activate a tool, the dialog bar displays commonly used options and collectors.
- Standard buttons for controlling the tool.
- Slide-down panels that open to reveal less commonly-used functionality. You can use them to perform advanced modeling actions or retrieve comprehensive feature information.
- A bitmap identifies the tool (typically the same icon used on buttons that invoke the tool).

Creo Parametric uses the dashboard to create features that involve extensive interaction with user interface components and geometry manipulation. You can use dashboards in Creo Parametric TOOLKIT applications:

- Where a dialog box is too large in size or is intrusive onto the graphics window. Dashboards enable you to make a smooth-flow tool.
- To present a streamlined "simple-user" activity with more complicated actions available to "expert users".
- Where Creo Parametric TOOLKIT dashboards are not only limited to feature creation activities and solid model modes.

Dashboard

An opaque pointer `ProUIDashboard` is a handle to the overall dashboard tool after it has been shown.

Showing a Dashboard

Functions Introduced:

- **ProUIDashboardShow()**
- **ProUIDashboardshowoptionsAlloc()**
- **ProUIDashboardshowoptionsNotificationSet()**
- **ProUIDashboardshowoptionsTitleSet()**
- **ProUIDashboardshowoptionsIconSet()**
- **ProUIDashboardshowoptionsHelpTextSet()**
- **ProUIDashboardshowoptionsDefaultOpenSet()**
- **ProUIDashboardshowoptionsFree()**
- **ProUIDashboardDestroy()**

Use the function `ProUIDashboardShow()` to push a new dashboard User Interface into the dashboard stack mechanism. The dashboard will be shown in the message area of Creo Parametric. This function creates an event loop, and thus does not exit until the dashboard is being dismissed.

Use the function `ProUIDashboardshowoptionsAlloc()` to allocate a handle containing data used to build a dashboard. The input arguments for this function are:

- *main_page*—Specifies the main page for the dashboard.
- *slideup_pages*—Specifies a `ProArray` of handles representing the slide-down pages, if needed.
- *appdata*—Specifies the application data to be stored with the dashboard.

Use the function `ProUIDashboardshowoptionsNotificationSet()` to assign a callback function to be called for the indicated event occurrence in the dashboard. The input arguments for this function are:

- *options*—Specifies a handle to data used to build a dashboard.
- *notification*—Specifies the notification function to be called for the given event.
- *appdata*—Specifies the application data to be passed to the callback function when it is invoked.

You can register event notifications for the following events on the dashboard:

- `PRO_UI_DASHBOARD_CREATE`—when the dashboard is first initialized (before the pages are initialized).
- `PRO_UI_DASHBOARD_SHOW`—when the dashboard is shown or resumed.
- `PRO_UI_DASHBOARD_HIDE`—when the dashboard is paused and replaced by another tool.

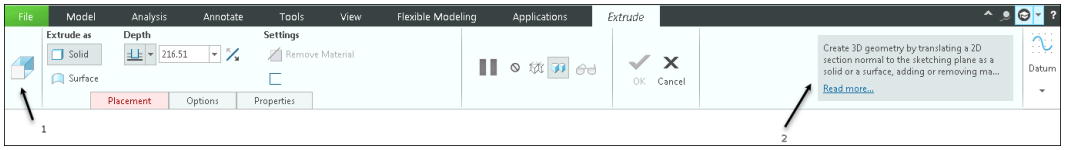
- PRO_UI_DASHBOARD_ENTER—when you switch to a dashboard from another component in the ribbon user interface
- PRO_UI_DASHBOARD_EXIT—when you leave the dash board and return to a component on the ribbon user interface
- PRO_UI_DASHBOARD_DISMISS—upon the dashboard “close” event.
- PRO_UI_DASHBOARD_DESTROY—when the dashboard is finally destroyed.

The function ProUIDashboardshowoptionsTitleSet() sets the title of the dashboard. The input arguments of this function are:

- *dash_options*—Specifies the handle containing data that is used to build the dashboard.
- *title*—Specifies the title of the dashboard.

Use the function ProUIDashboardshowoptionsIconSet() to set an icon for the specified dashboard. This is the first icon in the first group from the left on the dashboard.

The function ProUIDashboardshowoptionsHelpTextSet() sets the help text for the specified dashboard.



- 1. Dashboard icon
- 2. Dashboard help text

Use the function ProUIDashboardshowoptionsDefaultOpenSet() to set the specified dashboard as the open by default page. The input arguments follow:

- *dash_options*—Specify the handle to the data used to build the dashboard.
- *page_name*—Specify the page name to be opened by default.

You can use the function ProUIDashboardshowoptionsFree() to free a handle containing the data used to build a dashboard.

Use the function ProUIDashboardDestroy() to pop the dashboard from the dashboard stack mechanism. The dashboard User Interface will be destroyed.

Accessing a Dashboard

Functions Introduced:

- **ProUIDashboardUserdataGet()**
- **ProUIDashboardStdlayoutGet()**
- **ProUIDashboardBitmapSet()**

Use the functions `ProUIDashboardUserdataGet()` to get the application data you attached to the dashboard items upon registration. This can be used to store the current state of the tool, and thus to control the visibility of components.

Creo Parametric TOOLKIT does not currently provide access to create the feature dashboard's standard buttons. However, you can use `ProUIDashboardStdlayoutGet()` to get the layout name where you can create and place the buttons in the standard button area. Typically this consists of (at least) **OK** and **Cancel** buttons.

Use the function `ProUIDashboardBitmapSet()` to assign the icon for the dashboard, which appears to the left of the slide-down panel buttons.

Dashboard Page

Each section of content in a dashboard is called a dashboard page. The opaque handle `ProUIDashboardPage` represents an individual page, that is, either the dialog bar, or a single slide-down panel.

Dashboard Page Options

Functions Introduced:

- **ProUIDashboardpageoptionsAlloc()**
- **ProUIDashboardpageoptionsNotificationSet()**
- **ProUIDashboardpageoptionsFree()**

Use the function `ProUIDashboardpageoptionsAlloc()` to allocate a handle representing a single page (or layout) that will be shown in a dashboard. The input arguments for this function are:

- *page_name*— Specifies the page name (must be unique).
- *resource_name*—Specifies the name of the resource file to use (whose top component must be a layout, not a dialog). If `NULL`, an empty default layout is used.
- *application_data*—Specifies the application data stored for the page.

Use the function `ProUIDashboardpageoptionsNotificationSet()` to assign a function to be called upon a certain event occurring in the dashboard. The input arguments for this function are:

- *options*—Specifies a handle representing a dashboard page.
- *notification*—Specifies the function to be called upon the designated event occurrence.
- *appdata*—Specifies the application data passed to the callback function when it is invoked.

You can register event notifications for the following events on the dashboard page:

- `PRO_UI_DASHBOARD_PAGE_CREATE`—when the page is first created.
- `PRO_UI_DASHBOARD_PAGE_SHOW`—when the page is shown (slide-down panels only).
- `PRO_UI_DASHBOARD_PAGE_HIDE`—when the page is hidden (slide-down panels only).
- `PRO_UI_DASHBOARD_PAGE_DESTROY`—when the page is destroyed.

You can use the function `ProUIDashboardpageoptionsFree()` to free a handle representing a single page (or layout) that will be shown in a dashboard.

Accessing a Dashboard Page

Functions Introduced:

- **ProUIDashboardPageGet()**
- **ProUIDashboardpageTitleSet()**
- **ProUIDashboardpageForegroundcolorSet()**
- **ProUIDashboardStdlayoutDefaultBtnsAdd()**
- **ProUIDashboardStdlayoutButtonAdd()**
- **ProUIDashboardStdlayoutDefaultButtonNameGet()**
- **ProUIDashboardPauseresumeButtonStateGet()**
- **ProUIDashboardPauseresumeButtonStateSet()**
- **ProUIDashboardpageStateSet()**
- **ProUIDashboardpageNameGet()**
- **ProUIDashboardpageDashboardGet()**
- **ProUIDashboardpageUserdataGet()**
- **ProUIDashboardpageClose()**

Use the function `ProUIDashboardPageGet()` to obtain the handle to a given page from the dashboard. The input arguments for this function are:

- *dashboard*—Specifies the dashboard handle.
- *name*—Specifies the page name. Pass `NULL` to get the handle to the main page.

Use the function `ProUIDashboardpageTitleSet()` to assign the title string for the dashboard page. This will be shown as the button name for the slide-down panel. This should typically be called from the CREATE notification of the dashboard page.

Use the function `ProUIDashboardpageForegroundcolorSet()` to set the text color for the button that invokes a slide-down panel. This technique is used in several Creo Parametric tools to notify the user that they must enter one of the panels to complete the tool.

The function `ProUIDashboardStdlayoutDefaultBtnsAdd()` adds new standard push buttons to the Creo Parametric dashboard. The input arguments follow:

- *pageHandler*—A handle to the dashboard page.
- *buttons*—The bit mask to identify the buttons to be added. This value is defined by the enumerated data type `ProUIDashboardStdLayoutButton`. The valid values are as follows:
  - `PRO_UI_DASHBOARD_BUTTON_PAUSE_RESUME`
  - `PRO_UI_DASHBOARD_BUTTON_PREVIEW`
  - `PRO_UI_DASHBOARD_BUTTON_OK`
  - `PRO_UI_DASHBOARD_BUTTON_CANCEL`

The function `ProUIDashboardStdlayoutButtonAdd()` adds a new push button to the Creo Parametric dashboard. This function is executed only once during a Creo Parametric session for each push button. Subsequent calls to this function for a previously loaded push button are ignored. The input arguments follow:

- *page_handle*—A handle to the dashboard page.
- *button_Name*—A unique name for the push button. The maximum size should be less than `PRO_NAME_SIZE`.
- *button_label*—A label for the push button. The maximum size should be less than `PRO_LINE_SIZE`.
- *one_line_help*—A one-line help for the push button. The maximum size should be less than `PRO_LINE_SIZE`
- *icon*—An image of the push button.
- *filename*—The name of the message file that contains the label and help string.

`PRO_NAME_SIZE` and `PRO_LINE_SIZE` are defined in the `ProSizeConst.h` header file.

The function `ProUIDashboardStdlayoutDefaultButtonNameGet()` returns the default name of the specified button id. The output argument `button_name` is in the form of a character string. Use the function `ProStringFree()` to free this string.

The function `ProUIDashboardPauseresumeButtonStateGet()` returns the state of the button. The output argument `state` is defined by the enumerated data type `ProUIDashboardPauseResumeButtonState` and the valid are as values follows:

- `ProUIDashboardButtonPauseState`—Specifies that the button is in a paused state.
- `ProUIDashboardButtonResumeState`—Specifies that the button is in a resume state.

Use the function `ProUIDashboardPauseresumeButtonStateSet()` to set the pause or resume state of the button.

Use the function `ProUIDashboardpageStateSet()` to modify the visibility of the button that opens the dashboard page according to the page state. This function affects the background and foreground of the button. The input arguments follow:

- *page*—Handle to the dashboard page defined by `ProUIDashboardPage`.
- *state*—State of the page defined by the enumerated data type `ProUIDashboardPageState`. The valid values are as follows:
  - `PRO_UI_DASHBOARD_PAGE_DEFAULT_STATE`
  - `PRO_UI_DASHBOARD_PAGE_WARNING_STATE`
  - `PRO_UI_DASHBOARD_PAGE_ERROR_STATE`

Use the function `ProUIDashboardpageNameGet()` to obtain the name of the page.

Use the function `ProUIDashboardpageDashboardGet()` to obtain the dashboard that owns this page.

Use the function `ProUIDashboardpageUserdataGet()`to obtain the application stored with this dashboard page on registration.

Use the function `ProUIDashboardpageClose()` to close the dashboard slide-down page.

Accessing Components in the Dashboard Pages

Components added to the dashboard are actually "owned" by the Creo Parametric dialog window. Creo Parametric automatically modifies the names of components loaded from resource files to ensure that no name collisions occur when the components are added. The functions in this section allow you to locate the names of components that you need to access.

Functions Introduced:

- **ProUIDashboardpageDevicenameGet()**
- **ProUIDashboardpageComponentnameGet()**

Use the function `ProUIDashboardpageDevicenameGet()` to obtain the device name owning the dashboard page. This name should be used in other `ProUI` functions to access the components stored in the dashboard page.

Use the function `ProUIDashboardpageComponentnameGet()` to obtain the real component name in the dashboard page, if the page contents were loaded from a layout. This name should be used in `ProUI` functions for accessing a component in the dashboard.

20

User Interface: Basic Graphics

This chapter describes all the functions provided by Creo Parametric TOOLKIT that create and manipulate graphics and object displays.

Creo Parametric TOOLKIT refers to windows using integer identifiers. The base window (the big graphics window created automatically when you enter Creo Parametric) is window 0, and the text message window at the bottom is window 1.

In many of the functions in this section, you can use the identifier "–1" to refer to the current window (the one current to the Creo Parametric user).

Manipulating Windows

This section describes how to manipulate windows using Creo Parametric TOOLKIT . It is divided into the following subsections:

Windows

Functions Introduced:

- **ProWindowNameGet()**

The function `ProWindowNameGet()` returns the window name for the specified window identifier. The input parameter *win_id* is the identifier of the Creo Parametric window. The output parameter *win_name* is the name of the Creo Parametric window.

Resizing Windows

Functions Introduced:

- **ProWindowSizeGet()**
- **ProGraphicWindowSizeGet()**
- **ProWindowPixelOutlineGet()**
- **ProWindowCoordinatePixelGet()**

The function `ProWindowSizeGet()` returns the size of the Creo Parametric window including the User Interface border.

The function `ProGraphicWindowSizeGet()` returns the size of the Creo Parametric graphics window without the border. If the window occupies the whole screen. the window size is returned as 1. If the screen is 1024 pixels wide and the window is 512 pixels, the width will be returned as 0.5.

The function `ProWindowPixelOutlineGet()` returns the outline of the Creo Parametric window in pixels. The outline is the height and width of the graphic area.

The function `ProWindowCoordinatePixelGet()` converts the Windows coordinates received through the input argument *point* into Pixel coordinates. The input arguments are as follows:

- *window_id*—Valid window identifier.
- *point*—Window space 3D coordinates given by `ProArray` of the `Pro3dPnt` object.

The output argument *pixelSpaceCoord* is the 2D pixel coordinates in integers and is returned by a `ProArray` of the `Pro2dPnt` object.

Manipulating the Embedded Browser in Windows

Functions Introduced:

- **ProWindowBrowserSizeGet()**
- **ProWindowBrowserSizeSet()**
- **ProWindowURLShow()**
- **ProWindowURLGet()**

The functions `ProWindowBrowserSizeGet()` and `ProWindowBrowserSizeSet()` enable you to find and change size of the embedded browser in the Creo Parametric window. These functions refer to the browser size in terms of a percentage of the graphics window (0.0 to 100.0).

Note

The functions `ProWindowBrowserSizeGet()` and `ProWindowBrowserSizeSet()` are not supported if the browser is open in a separate window.

The functions `ProWindowURLGet()` and `ProWindowURLShow()` enable you to find and change the URL displayed in the embedded browser in the Creo Parametric window.

Repainting Windows

Functions Introduced:

- **ProWindowClear()**
- **ProWindowRepaint()**
- **ProWindowRefresh()**
- **ProWindowRefit()**
- **ProTreetoolRefresh()**

The function `ProWindowClear()` temporarily removes all graphics from the specified window. If you give the function a window identifier of –1, it clears the current window.

This function is not equivalent to the Creo Parametric option to quit the window. It does not break the connection between the current solid and the window. That is the purpose of the function `ProWindowDelete()`, described later in this chapter.

The `ProWindowRepaint()` function is equivalent to the Creo Parametric command **Repaint** in the **Graphics** toolbar, and removes highlights. The function accepts –1 as the identifier, which indicates the current view.

The function `ProWindowRefresh()` is designed primarily for the purposes of animation. It updates the display very efficiently, but does not remove highlights. The function accepts –1 as the window identifier, which indicates the current window.

The function `ProWindowRefit()` performs exactly the same action as the Creo Parametric command **View ► Refit** . (It does not accept -1 as the current window. Use `ProWindowCurrentGet()` if you need the id of the current window.)

The function `ProTreetoolRefresh()` refreshes the display of the model tree for the specified model. This function is useful when you are modifying the model in some way, such as when you are creating patterns or features.

Controlling Which Window is Current

Functions Introduced:

- **ProWindowCurrentGet()**
- **ProWindowCurrentSet()**
- **ProAccessorywindowAboveactivewindowSet()**

The functions `ProWindowCurrentGet()` and `ProWindowCurrentSet()` enable you to find out and change the current window. The window is active only for the purposes of the other Creo Parametric TOOLKIT commands that affect windows.

The function `ProWindowCurrentSet()` is not equivalent to the Creo Parametric command to activate the window and has no effect on the object returned by `ProMdlCurrentGet()`.

The function `ProAccessorywindowAboveactivewindowSet()` allows you to display the accessory window always above the active Creo Parametric window, for the current Creo Parametric TOOLKIT application. Pass the input argument `above_active_window` as `PRO_B_TRUE` to display the accessory window on top of the current window. To remove this setting, pass the input argument `above_active_window` as `PRO_B_FALSE`.

Note

The configuration option `accessory_window_above` allows you to control the display of accessory window in Creo Parametric. The valid values are:

- *yes*—Always displays the accessory window above the active window.
- *no*—Does not display the accessory window above the active window. Here, whichever is the active window is displayed on top.

Creating and Removing Windows

Functions Introduced:

- **ProObjectwindowMdlnameCreate()**
- **ProWindowDelete()**
- **ProWindowCurrentClose()**
- **ProAccessorywindowWithTreeMdlnameCreate()**

The function `ProObjectwindowMdlnameCreate()` opens a new window containing a specified solid. The solid must already be in memory. If a window is already open on that solid, the function returns the identifier of that window. If the Main Window is empty, the function uses it instead of creating a new one. The section Graphics Colors and Line Styles on page 486 shows how to use `ProObjectwindowMdlnameCreate()`.

The function `ProWindowDelete()` closes a window and breaks the object-to-window attachment. The function deletes the window, if it is not the base window. You cannot break the attachment for the currently active window. Use the function `ProWindowCurrentSet()` to make a different window be the current window before calling this function.

The function `ProWindowDelete()` is the equivalent of the Creo Parametric command to quit the window. If the window is not the Main Window, it is also deleted from the screen.

To close the current window, use the function `ProWindowCurrentClose()`. When you call this function, the control must be returned to Creo Parametric to close the current window.

This function duplicates the behavior of the **View ▸ Close** command in Creo Parametric. If the current window is the original window created when Creo Parametric was started, the function clears the window; otherwise, the function removes the window from the screen.

Note

Any work done since the last save will be lost.

In Creo Parametric, when the main window is active, you can open an accessory window for operations such as edit an inserted component or a feature, preview an object, select a reference, and some other operations. The model tree associated with this Creo Parametric object is also displayed in the accessory window. For more information about the accessory window, refer the Creo Parametric help.

In Creo Parametric TOOLKIT , the function `ProAccessorywindowWithTreeMdlnameCreate()` opens an accessory window containing the specified object. If a window is already open with the specified object, the function returns the identifier of that window. If an empty window exists, the function uses that window to open the object. The input argument *tree_flag* controls the display of the model tree in the accessory window. If this flag is set to `PRO_B_TRUE` the model tree is displayed.

Retrieving the Owner of a Window

Function Introduced:

- **ProWindowMdlGet()**

The function `ProWindowMdlGet()` retrieves the Creo Parametric model that owns the specified window. This function gives you details about the window needed to perform necessary actions on it.

Note

If no model is associated with the specified window, Creo Parametric TOOLKIT returns `NULL` as a model pointer and `PRO_TK_NO_ERROR` as a return value.

Visiting Windows

Function Introduced:

- **ProWindowsVisit()**

The function `ProWindowsVisit()` enables you to visit all the Creo Parametric windows. For a detailed explanation of visiting functions, see the section Visit Functions in the Fundamentals on page 22 chapter.

Activating Windows

Function Introduced:

- **ProWindowActivate()**

The function `ProWindowActivate()` activates the specified window and sets it as the current window. When you call this function, the control must be returned to Creo Parametric to activate the specified window.

You can regain control by registering callback using the function `ProUIDialogAppActionSet()`.

This functionality is equivalent to changing the active window by selecting and activating a window using the pull-down menu of **Windows** command under the **View** tab in Creo Parametric.

Note

This function works in asynchronous graphics mode only.

Flushing the Display Commands to Window

Function Introduced:

- **ProWindowDeviceFlush()**

When an application sends commands to display graphics, these calls are buffered. The buffered commands are executed whenever there is a movement of mouse in the graphics window. If the Creo Parametric TOOLKIT application modifies the current display, but does not have any interaction with the graphics window, the function `ProWindowDeviceFlush()` must be called. The function flushes the buffers and executes all the display commands on the specified window. For example, consider a link which is clicked in an embedded browser. This link alters the display in the graphics window. As there is no movement of the mouse in the graphics window, the function `ProWindowDeviceFlush()` must be called to execute the display calls.

Note

You must not call this function often, as it causes the systems running on Windows Vista and Windows 7 to slow down. It is recommended that you call this function only after you complete all the display operations.

Solid Orientation

Functions Introduced:

- **ProWindowCurrentMatrixGet()**
- **ProViewMatrixGet()**
- **ProViewMatrixSet()**
- **ProViewReset()**
- **ProViewRotate()**
- **ProWindowPanZoomMatrixSet()**
- **ProViewRefit()**

Each graphics window in solid (Part or Assembly) mode has two transformation matrices associated with it—the view matrix and the window matrix. The view matrix describes the transformation between solid coordinates and screen coordinates. Therefore, the view matrix describes the orientation of the solid in the window.

The window matrix is the transformation between screen coordinates and window coordinates. The window matrix describes the pan and zoom factors. The screen coordinate at which a particular point on a solid is displayed is not affected by pans and zooms—this affects window coordinates only.

The view matrix is important because the mouse input functions and some of the graphics drawing functions use screen coordinates, while all solid geometry uses solid coordinates. The view matrix enables you to transform between the two systems. The function `ProWindowCurrentMatrixGet()` provides the window matrix for the current window.

The function `ProWindowPanZoomMatrixSet()` enables you to set the pan and zoom matrix (window matrix) for the current window.

The function `ProViewRefit()` zooms and pans the view to display the specified object in the window. The input arguments are:

- *model*—Handle to the object. The supported object types are drawing, part, and assembly.
- *view*—Handle to the view, which is used to display the object. If the object is a solid model, and is displayed in the current window, you can pass the argument as `NULL`.

  If the object is a drawing, pass the handle to the background view. Use the function `ProDrawingBackgroundViewGet()` to get the handle to the background view.

Getting and Setting the View Matrix

The function `ProViewMatrixGet()` provides the view matrix for a specified window. Set the view argument to `NULL` for the current view.

The function `ProViewMatrixSet()` enables you to set the view matrix (if normalized), and therefore the orientation of the solid in the view.

Note

Function `ProViewMatrixSet()` does not cause the view to be repainted.

A 4x4 transformation matrix describes a shift and a scaling, as well as a reorientation. You set the view matrix to define a new orientation. Creo Parametric applies its own shift and scaling to the view matrix you provide to ensure that the solid fits properly into the view. This implies the following:

- The matrix output by `ProViewMatrixGet()` is not the same as the one you previously input to the function `ProViewMatrixSet()`, although its orientation is the same.
- Each row of the matrix you provide to `ProViewMatrixSet()` must have a length of 1.0, and the bottom row must be 0, 0, 0, 1.
- The matrix you provide to `ProViewMatrixSet()` must be normalized—it cannot include scaling or shift. Example 1: Saving Three Views on page 502 shows how to normalize a matrix.

Converting a Matrix to Orthonormal

Functions Introduced:

- **ProMatrixMakeOrthonormal()**

The function `ProMatrixMakeOrthonormal()` converts a non-orthonormal matrix to an orthonormal matrix with the specified scaling factor.

The input arguments follow:

- *inMatrix*—The matrix to be converted to orthonormal.
- *intended_scale*—Scale factor to be applied on the matrix.

Storing Named Views

Functions Introduced:

- **ProViewStore()**
- **ProViewRetrieve()**

- **ProViewNamesGet()**
- **ProViewFromModelitemGet()**
- **ProViewNameLineGet()**
- **ProViewIdFromNameLineGet()**
- **ProViewNameSet()**
- **ProViewDelete()**

The `ProViewStore()` and `ProViewRetrieve()` functions enable you to save and use a named view of the solid. They are equivalent to the Creo Parametric **View ► Reorient** commands **Save**, and **Set** in the **Saved Views** tab, under the **Orientation** dialog box. You can then select the view you want from the list of view names.

The function `ProViewNamesGet()` retrieves the names of the views in the specified solid.

The function `ProViewFromModelitemGet()` retrieves the view handle from a model item handle. The model item must be of type `PRO_VIEW`.

The function `ProViewNameGet()` has been superseded by the function `ProViewNameLineGet()`. The function `ProViewNameLineGet()` retrieves the name of the view from the view handle.

The function `ProViewIdFromNameGet()` has been superseded by the function `ProViewIdFromNameLineGet()`. The function `ProViewIdFromNameLineGet()` retrieves the ID of the view. The input arguments are:

- *model*—Specifies the handle to the part or assembly associated with the drawing, or to the drawing that contains the view. This argument cannot be `NULL`.
- *view_name*—Specifies the name of the view. This argument cannot be `NULL`.

The function `ProViewNameSet()` sets the name of the view in the specified solid. The inputs arguments are:

- *model*—Specifies the handle to a part, assembly, or drawing. This argument cannot be `NULL`.
- *p_view*—Specifies the handle of the view.
- *p_name*—Specifies the name of the view.

The function `ProViewDelete()` deletes the view from the specified solid. The input arguments are:

- *model*—Specifies the handle to a part, assembly, drawing. This argument cannot be `NULL`.
- *p_view*—Specifies the handle of the view.

Example 1: Saving Three Views

The sample code in `UgGraphViewsSave.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` shows how to set the view matrix and store a named view.

Graphics Colors and Line Styles

Creo Parametric uses several predefined colors in its color map. The colors are represented by the values of `ProColorType()`, defined in the file `ProToolkit.h`. The names of the types generally indicate what they are used for in Creo Parametric, although many colormap entries are used for several different purposes. These also correspond to the system colors presented to the user through the user interface.

Note

PTC reserves the right to change both the definitions of the predefined colormap and also of the assignment of entities to members of the color map as required by improvements to the user interface. PTC recommends not relying on the predefined RGB color for displaying of Creo Parametric TOOLKIT entities or graphics, and also recommends against relying on the relationship between certain colormap entries and types of entities. The following section describe how to construct your application so that it does not rely on potentially variant properties in Creo Parametric.

Setting Colors to Desired Values

Functions Introduced:

- **ProTextColorModify()**
- **ProGraphicsColorModify()**

The functions `ProTextColorModify()` and `ProGraphicsColorModify()` enable you to select a different color to be used for either of the following:

- Graphics text—User custom text drawn by the `ProGraphicsTextDisplay`. Graphics text is by default displayed using `PRO_COLOR_LETTER`.
- Graphics—User custom graphics drawn by `ProGraphics` function, which is by default displayed using `PRO_COLOR_DRAWING`.

Both functions only affect the color used for new graphics, you draw using Creo Parametric TOOLKIT , not the colors used for items Creo Parametric draws.

Both functions take a ProColor structure as input. This structure allows you to specify color by one of the following three methods:

- DEFAULT—use the default Creo Parametric color entry for new graphics or text.
- TYPE—use a predefined `ProColortype` color.
- RGB - use a custom RGB value. This method should be used for any graphics which should not be allowed to change color (for example; if an application wants a yellow line on the screen that should always be yellow and not depend on the chosen color scheme.

Both functions output the value of the previous setting. It is good practice to return the color to its previous value after having finished drawing an object.

Setting Colors to Match Existing Entities

Functions Introduced:

- **ProColorByTypeGet()**

The functions `ProGraphicsColorModify()` and `ProColorTextModify()` allow you to draw graphics that will change color based on changes to the Creo Parametric colormap. This allows you to draw entities in similar colors to related entities created by Creo Parametric. However, if the associations between objects and colormap entries should change in a new release of Creo Parametric, the association between the application entities and the Creo Parametric entities would be lost. The function `ProColorByTypeGet()` returns the standard colormap entry corresponding to a particular entity in Creo Parametric. This allows applications to draw graphics that will always match the color of a particular Creo Parametric entity.

Example 2: Setting the Graphics Color to a Specific RGB Value

The sample code in `UgGraphColorsAdjust.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` shows how to set the graphics color to a specific RGB value.

Example 3: Setting The Graphics Color to Follow the Color of Creo Parametric Entity

The sample code in `UgGraphColorsAdjust.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` shows how to set the graphics color to follow the color of Creo Parametric Entity.

Modifying the Creo Parametric Color Map

Functions Introduced:

- **ProColormapGet()**
- **ProColormapSet()**

These functions enable you to find out and alter the color settings for Creo Parametric. Each color is defined in terms of the red, green, and blue values. The RGB values should be expressed in a range from 0.0 to 1.0. Note that some colors related to selection and highlighting are fixed and may not be modified.

Creo Parametric uses these colors for everything it displays.

Changes to the color map are preserved for the rest of the Creo Parametric session. If you want to have permanent changes, call `ProColormapSet()` in `user_initialize()`.

Note

Changing the Creo Parametric color map can have unintended effects if the user has chosen an alternate color scheme. It may cause certain entries to blend into the background or to be confused with other types of entries.

Example 4: Modifying the Color of the HALF_TONE Display

The sample code in `UgGraphColorsAdjust.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` shows how to modify the default half-tone color (gray) to a brighter color, using the function `ProColormapSet()`. Creo Parametric uses half-tone color to display hidden lines.

Creo Parametric Color Schemes

Functions Introduced:

- **ProColormapAlternateschemeGet()**
- **ProColormapAlternateschemeSet()**

The functions `ProColormapAlternateschemeGet()` and `ProColormapAlternateschemeSet()` enable you to change the color scheme of Creo Parametric to a predefined color scheme by turning on and off the blended background (light to dark grey) for alternate schemes. The possible alternate color schemes are as follows:

- `PRO_COLORMAP_ALT_BLACK_ON_WHITE`—Displays black entities shown on a white background.
- `PRO_COLORMAP_ALT_WHITE_ON_BLACK`—Displays white entities shown on a black background.
- `PRO_COLORMAP_ALT_WHITE_ON_GREEN`—Displays white entities shown on a dark green background.
- `PRO_COLORMAP_OPTIONAL1`—Represents the color scheme with a dark background.
- `PRO_COLORMAP_OPTIONAL2`—Represents the color scheme with a medium background.
- `PRO_COLORMAP_CLASSIC_WF`—Resets the color scheme to the light to dark grey background (default upto Pro/ENGINEER Wildfire 4.0).
- `PRO_COLORMAP_ALT_DEFAULT`—Resets the color scheme to the default color scheme of light to dark blue gradient background (from Pro/ENGINEER Wildfire 5.0 onwards).

Setting Line Styles for Creo Parametric TOOLKIT Graphics

Functions Introduced:

- **ProLinestyleSet()**
- **ProLinestyleDataGet()**
- **ProGraphicsModeSet()**

The function `ProLineStyleSet()` enables you to set the style of graphics you draw. The function `ProLinestyleDataGet()` queries the definition of the line style.

The possible values for the line style are as follows:

- `PRO_LINESTYLE_SOLID`—Solid line
- `PRO_LINESTYLE_DOT`—Dotted line
- `PRO_LINESTYLE_CENTERLINE`—Alternating long and short dashes
- `PRO_LINESTYLE_PHANTOM`—Alternating long dashes and two dots

Displaying the Color Selection Dialog Box

Function Introduced:

- `ProUIColorSelectionShow()`

The function `ProUIColorSelectionShow()` displays the dialog box used to select values for the red, green, blue (RGB) colors. The input arguments of this function are:

- `title` – Specifies the title of the selection dialog box. If this argument is NULL, the default value RGB will be used.
- `default_rgb_color` – Specifies the default RGB values that will be displayed when the dialog box is opened. The color black is selected, if the value specified for this argument is invalid.

Displaying Graphics

Functions Introduced:

- **ProGraphicsPenPosition()**
- **ProGraphicsLineDraw()**
- **ProGraphicsPolylineDraw()**
- **ProGraphicsMultiPolylinesDraw()**
- **ProGraphicsArcDraw()**
- **ProGraphicsCircleDraw()**
- **ProGraphicsPolygonDraw()**

All the functions in this section draw graphics in the current window (the Creo Parametric current window, unless redefined by a call to `ProWindowCurrentSet()`), and use the color and line style set by calls to `ProGraphicsColorSet()` and `ProLinestyleSet()`. The functions draw the graphics in the Creo Parametric graphics color. The default graphics color is white.

By default, the graphics elements are not stored in the Creo Parametric display list, so they do not get redrawn by Creo Parametric when the user selects the **Repaint** command or the orientation commands in the **Orientation** group in the **View** tab. However, if you store graphics elements in either 2-D or 3-D display lists, Creo Parametric redraws them. See the section Display Lists on page 493 for more information.

The functions `ProGraphicsPenPosition()` and `ProGraphicsLineDraw()` draw three-dimensional polylines in solid mode, and take solid coordinates.

The function `ProGraphicsPenPosition()` sets the point at which you want to start drawing the line. The function `ProGraphicsLineDraw()` draws a line to the given point from the position given in the last call to either of the two functions. You call `ProGraphicsPenPosition()` for the start of the polyline, and `ProGraphicsLineDraw()` for each vertex.

If you use these functions in Drawing mode they work correctly, but use screen coordinates instead of solid coordinates.

The `ProGraphicsPolylineDraw()` and `ProGraphicsMultiPolylinesDraw()` functions also draw polylines, but you need to have the whole polyline defined in a local array before you call either function. If you are drawing many lines, use `ProGraphicsMultipolylinesDraw()` to minimize the number of function calls.

The function `ProGraphicsArcDraw()` draws an arc, in screen coordinates.

The function `ProGraphicsCircleDraw()` uses solid coordinates for the center of the circle and the radius value, but draws the circle parallel to the plane of the window. You can position the circle at a chosen solid vertex, for example, and the circle will always be clearly visible as a circle, regardless of the current solid orientation.

Example 5: Displaying Lines and Circles

The sample code in `UgGraphLineCircleDraw.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` shows how to draws a circle at each point and draws lines between the points.

In this example, the user selects a series of points on a part surface.

Displaying Text

Function Introduced:

- **ProGraphicsTextDisplay()**

The function `ProGraphicsTextDisplay()` places text, specified as a wide string, at a position specified in screen coordinates. Therefore, if you want to add text to a particular position on the solid, you must transform the solid coordinates into screen coordinates by using the view matrix.

Like the graphics polylines, arcs, and so on (added by the functions described in the section Displaying Graphics on page 490), the text items drawn by `ProGraphicsTextDisplay()` are not known to Creo Parametric, and therefore are not redrawn when you use the **Repaint** command. Use the notify or display list functions to tell Creo Parametric about the items. See the section

Display Lists on page 493 for more information on the display list functions. To add permanent text to a drawing (for example, a drawing note), see the section Drawings on page 1226.

Controlling Text Attributes

Functions Introduced:

- **ProTextAttributesCurrentGet()**
- **ProTextFontIdCurrentSet()**
- **ProTextHeightCurrentSet()**
- **ProTextRotationAngleCurrentSet()**
- **ProTextSlantAngleCurrentSet()**
- **ProTextWidthFactorCurrentSet()**

These functions control the attributes of text added by calls to `ProGraphicsTextDisplay()`. You can get and set the following information:

- The font identifier
- The text height, in screen coordinates
- The ratio of the width of each character (including the gap) as a proportion of the height
- The angle of rotation of the whole text, in counterclockwise degrees
- The angle of slant of the text, in clockwise degrees

Controlling Text Fonts

Functions Introduced:

- **ProTextFontDefaultIdGet()**
- **ProTextFontNameGet()**
- **ProTextFontNameToId()**
- **ProTextFontRetrieve()**
- **ProTextStyleFontGet()**
- **ProTextStyleFontSet()**

The function `ProTextFontDefaultIdGet()` returns the identifier of the default Creo Parametric text font.

The text fonts are identified in Creo Parametric by names, and in Creo Parametric TOOLKIT by integer identifiers. To move between the two types of font identifiers, use the functions `ProTextFontNameGet()` and

`ProTextFontNameToId()`. Because the internal font identifiers could change between Creo Parametric sessions, it is important to call `ProTextFontNameToId()` each time you want to modify the font in Creo Parametric TOOLKIT.

The function `ProTextFontRetrieve()` loads a font with the specified name that can be used to display the text.

The functions `ProTextStyleFontGet()` and `ProTextStyleFontSet()` get and set the font used to display the text. The fonts are those available in the **Font** selector in the **Text Style** dialog in Creo Parametric.

Validating Text Styles

Functions Introduced:

- **ProTextStyleValidate()**

The function `ProTextStyleValidate()` checks whether the properties of text style applied to the specified object type are valid. The input arguments are:

- *obj_type*—Specifies the type of object on which the text style must be applied. The enumerated data type `ProTextStyleObjectType` defines the object type.
- *r_text_style*—Specifies the text style that should be applied on the specified object type.

The function returns messages if it finds text styles which are not supported in the specified object type. The message is returned as array of lines. Use the function `ProWstringArrayFree()` to free memory.

Display Lists

Functions Introduced:

- **ProDisplist2dCreate()**
- **ProDisplist2dDisplay()**
- **ProDisplist2dDelete()**
- **ProDisplist3dCreate()**
- **ProDisplist3dDisplay()**
- **ProDisplist3dDelete()**

To generate the display of a solid in a window, Creo Parametric maintains two display lists. A display list contains a set of vectors used to represent the shape of the solid in the view.

The 3-D display list contains a set of three-dimensional vectors that represent an approximation to the geometry of the edges of the solid. It gets rebuilt every time the solid is regenerated.

The 2-D display list contains the two-dimensional projections of the 3-D display list onto the current window. It is rebuilt from the 3-D display list when the orientation of the solid changes.

The functions in this section enable you to add your own vectors to the display lists, so your own graphics will be redisplayed automatically by Creo Parametric, until the display lists are rebuilt.

For example, if you just use the functions described in the section Displaying Graphics on page 490, the items you draw remain on the screen only until the window is repainted. Furthermore, the objects are not plotted with the rest of the contents of the window because Creo Parametric does not know about them.

If, however, you add the same graphics items to the 2-D display list, they will survive each repaint (when zooming and panning, for example) and will be included in plots created by Creo Parametric.

If you add graphics to the 3-D display list, you get the further benefit that the graphics survive a change to the orientation of the solid and are displayed, even when you spin the solid dynamically.

To add graphics to a display list, you must write a function that displays the necessary vectors in three dimensions, using the graphics display functions in the usual way. For the 2-D display list, you call `ProDisplist2dCreate()` to tell Creo Parametric to use your function to create the display list vectors, then call `ProDisplist2dDisplay()` to ask it to display the new graphics those vectors represent.

Note

If you save the display information, you can reuse it in any session. The application should delete the display list data when it is no longer needed.

Using 3-D display lists is exactly analogous to using 2-D display lists.

Note that the function `ProWindowRefresh()` does not cause either of the display lists to be regenerated, but simply repaints the window using the 2-D display list.

The function `ProSolidDisplay()` regenerates both display lists, and therefore not only recenters the solid in the view and removes any highlights, but also removes any graphics you added using the 2-D display list functions.

Getting Mouse Input

Functions Introduced:

- **ProMousePickGet()**
- **ProMouseTrack()**
- **ProMouseBoxInput()**

The functions `ProMousePickGet()` and `ProMouseTrack()` are used to read the position of the mouse, in screen coordinates. Each function outputs the position and an enumerated type description of which mouse buttons were pressed when the mouse was at that position. The values are defined in `ProGraphic.h`, and are as follows:

- `PRO_LEFT_BUTTON`
- `PRO_MIDDLE_BUTTON`
- `PRO_RIGHT_BUTTON`

The function`ProMousePickGet()` reports the mouse position only when you press a button. It has an input argument that is a description of the mouse buttons you want to wait for (you must have at least one). For example, you can set the *expected_button* argument to:

```
PRO_LEFT_BUTTON | PRO_RIGHT_BUTTON
```

In this example, the function does not return until you press either the left or right button. You could also specify the value `PRO_ANY_BUTTON`.

The function `ProMouseTrack()` returns whenever the mouse is moved, regardless of whether a button is pressed. Therefore, the function can return the value `PRO_NO_BUTTON`. Its input argument enables you to control whether the reported positions are snapped to grid.

Example 6: Drawing a Rubber-Band Line

The sample code in `UgGraphPolyLineDrawUtil.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_graphics` shows how to use the complement drawing mode. The example follows the mouse position dynamically.

Cosmetic Properties

You can enhance your model using Creo Parametric TOOLKIT functions that change the surface properties, or set different light sources. The following sections describe these functions in detail.

Surface Properties

Functions Introduced:

- **ProSurfaceSideAppearancepropsGet()**
- **ProSurfaceSideAppearancepropsSet()**
- **ProTexturePathGet()**
- **ProSurfaceSideTextureplacementpropsGet()**
- **ProSurfaceSideTextureplacementpropsSet()**
- **ProSurfaceAppearanceDefaultPropsGet()**
- **ProSurfaceSideTexturepropsGet()**
- **ProSurfaceSideTexturepropsSet()**
- **ProPartTessellate()**
- **ProPartTessellationFree()**

From Creo Parametric5.0.0.0 onwards, the following functions have been deprecated:

- `ProSurfaceAppearancepropsGet()`
- `ProSurfaceAppearancepropsSet()`
- `ProSurfaceTextureplacementpropsGet()`
- `ProSurfaceTextureplacementpropsSet()`
- `ProSurfaceTexturepropsGet()`
- `ProSurfaceTexturepropsSet()`

The functions described in this section enable you to retrieve and set the surface and texture properties for the first level models in the model hierarchy.

Use the functions `ProSurfaceSideAppearancepropsGet()` and `ProSurfaceSideAppearancepropsSet()` to retrieve and set the surface properties on a specified side of the surface for the specified element using the `ProSurfaceAppearanceProps` data structure.

The data structure is defined as follows:

```
typedef struct pro_surf_appearance_props
{
  double                ambient;
  double                diffuse;
  double                highlite;
  double                shininess;
  double                transparency;
  ProVector             color_rgb;
  ProVector             highlight_color;
  double                reflection;
  ProPath               name;
  ProPath               label;
```

```
  ProPath                description;
  ProPath               keywords;
} ProSurfaceAppearanceProps;
```

The `ProSurfaceAppearanceProps` data structure contains the following fields:

- `ambient`—Specifies the indirect, scattered light the model receives from its surroundings. The valid range is 0.0 to 1.0.
- `diffuse`—Specifies the reflected light that comes from directional, point, or spot lights. The valid range is 0.0 to 1.0.
- `highlite`—Specifies the intensity of the light reflected from a highlighted surface area. The valid range is 0.0 to 1.0.
- `shininess`—Specifies the properties of a highlighted surface area. A plastic model would have a lower shininess value, while a metallic model would have a higher value. The valid range is 0.0. to 1.0.
- `transparency`—Specifies the transparency value, which is between 0 (completely opaque) and 1.0 (completely transparent).
- `color_rgb[3]`—Specifies the color, in terms of red, green, and blue. The valid range is 0.0. to 1.0.
- `highlight_color`—Specifies the highlight color, in terms of red, green, and blue. The valid range is 0.0 to 1.0.
- `reflection`—Specifies how reflective the surface is. The valid range is 0 (dull) to 100 (shiny).
- `keywords`—Mandatory field. Specifies how to set the texture on a model surface. If you do not want to set a value for this field, set a `NULL` string to it.

Note

To set the default surface appearance properties, pass the argument *appearance_properties* as `NULL` in the `ProSurfaceAppearancepropsSet()` function.

The input arguments to the function `ProSurfaceSideAppearancepropsGet()` are:

- *item*—Specifies the `ProModelitem` object that represents the part, assembly component, subassembly, quilt, or surface.
- *surface_side*—Specifies the direction of the side for the surface or quilt. Pass the value as 0 to specify the side which is along the surface normal. Pass 1 to specify the side opposite to the surface normal.

In cosmetic shade mode, Creo Parametric tessellates each surface by breaking it into a set of connected triangular planes. The function `ProSurfaceTessellationGet()` invokes this algorithm on a single specified surface and provides the coordinates of the triangle corners and the normal at each vertex. This function tessellates a single surface only.

You can use the function `ProTexturePathGet()` to retrieve the full path to the texture, decal, or bump map file.

Refer to the Tessellation on page 181 section of Core: 3D Geometry on page 170 the chapter for functions `ProPartTessellate()` and `ProPartTessellationFree()`.

You can apply textures to the surfaces. Use the function `ProSurfaceSideTexturepropsGet()` to get the texture properties for the specified side of the surface. Use the function `ProSurfaceSideTexturepropsSet()` to set the surface texture properties for the specified side of the surface. Both these functions use the `ProSurfaceTextureProps` data structure to retrieve and set the surface texture properties.

The data structure is defined as follows:

```
typedef struct pro_surface_texture_props
{
  ProCharPath                          decal;
  ProCharPath                          texture_map;
  ProCharPath                          bump_map;
} ProSurfaceTextureProps;
```

The `ProSurfaceTextureProps` data structure contains the following fields:

- `decal`—Specifies the full path to the texture map with the alpha channel (transparency). Otherwise, use `NULL`.
- `texture_map`—Specifies the full path to the texture map.
- `bump_map`—Specifies the full path to the bump map. A bump map enables you to create bumps on the surface of the texture map.

You can manipulate the placement of the surface textures. Use the function `ProSurfaceSideTextureplacementpropsGet()` to get the properties related to the placement of surface texture for the specified side of the surface. Use the function `ProSurfaceSideTextureplacementpropsSet()` to set the placement properties for the specified side of the surface texture. The functions `ProSurfaceSideTextureplacementpropsGet()` and `ProSurfaceSideTextureplacementpropsSet()` use the `ProSurfaceTexturePlacementProps` data structure.

The data structure is defined as follows:

```
typedef struct pro_surface_texture_placement_props
{
```

```
  ProTextureProjectionType                    projection;
  ProTextureType                              texture_type;
  ProLineEnvelope                             local_csys;
  double                                      horizontal_offset;
  double                                      vertical_offset;
  double                                      rotate;
  double                                      horizontal_scale;
  double                                    vertical_scale;
  double                                    bump_height;
  double                                    decal_intensity;
  ProBoolean                                flip_horizontal;
  ProBoolean                                flip_vertical;
} ProSurfaceTexturePlacementProps;
```

The `ProSurfaceTexturePlacementProps` data structure contains the following fields:

- `projection`—Specifies the projection type—planar, spherical, cylindrical, or box.
- `texture_type`—Specifies the type of texture.
- `local_csys`—Specifies the direction (for planar projection), or the whole coordinate system (which defines the center for the other projection types).
- `horizontal_offset` and `vertical_offset`—Specifies the percentage of horizontal and vertical shift of the texture map on the surface.
- `rotate`—Specifies the angle to rotate the texture map on the surface.
- `horizontal_scale` and `vertical_scale`—Specifies the horizontal and vertical scaling of the texture map.
- `bump_height`—Specifies the height of the bump on the surface of the texture map.
- `decal_intensity`—Specifies the alpha or transparency mask intensity on the surface.
- `flip_horizontal` and `flip_vertical`—Specifies that the texture map on the surface should be flipped horizontally or vertically.

Use the function `ProSurfaceAppearanceDefaultPropsGet()` to get the default appearance properties of the specified type of surface. The input argument *appearance_type* is specified by the enumerated data type `ProDefaultAppearanceType`. The output argument `appearance_properties` is specified by the data structure `ProSurfaceAppearanceProps`.

Setting Light Sources

Functions Introduced:

- **ProLightSourcesGet()**
- **ProLightSourcesSet()**

These functions retrieve and set the information associated with the specified window, respectively, using the *ProLightInfo* data structure. The data structure is defined as follows:

```
typedef struct pro_tk_light
{
   wchar_t                                                  name
[PRO_NAME_SIZE];
   ProLightType                                             type;
   ProBoolean                                               status;
/* active or inactive */
   double                                                   rgb[3];
/* for all types */
   double                                                   position
[3]; /* for point and spot */
   double
direction[3]; /* for direction and spot */
   double                                                   spread_
angle; /* for spot, in radian */
   ProBoolean                                               cast_
shadows;
} ProLightInfo;
```

The `Pro_light` structure contains the following fields:

- `name`—Specifies the name of the light source.
- `type`—Specifies the light type—ambient, direction, point, or spot.
- `status`—Specifies whether the light source is active or inactive.
- `rgb`—Specifies the red, green, and blue values, regardless of the light type.
- `position`—Specifies the position of the light, for point and spot lights only.
- `direction`—Specifies the direction of the light, for direction and spot lights only.
- `spread_angle`—Specifies the angle the light is spread, for spot lights only.
- `cast_shadows`—Specifies whether the light casts shadows. This applies to Creo Render Studio only. Refer to the Creo Render Studio Help for more information on Creo Render Studio.

Creating 3D Shaded Data for Rendering

The functions described in this section enable you to create 3D shaded data that can be attached to a scene and rendered in Creo Parametric.

Functions Introduced:

- **ProDispObjectCreate()**
- **ProDispObjectAttach()**
- **ProDispObjectDetach()**
- **ProDispObjectDelete()**
- **ProDispObjectSetTransform()**
- **ProDispObjectSetSurfaceAppearanceProps()**

The function `ProDispObjectCreate()` creates a display object. A display object collects user specified shaded triangle data. When you render an object in Creo Parametric, rendering is done based on the triangle data using lights and materials. The display object must be attached to the scene to be rendered. The object is always displayed in shaded mode irrespective of the display mode of the current view. No HLR operations are done on display objects. The input arguments are:

- *object_name*—Specifies the name of the display object. The name must be unique.
- *flag*—Specifies a bitmask that is used to set the property of the display object. The bitmask has the following bit flags:
  - `0x0`—This is the default value. No bit flag is passed. If no bit is set, the display object behaves like a solid.
  - `PRO_DISP_OBJECT_TWO_SIDED`—When this bit flag is set, the display object behaves like a quilt.
  - `PRO_DISP_OBJECT_DYNAMIC_PREVIEW`—When this bit flag is set, the display object is considered temporary which will be deleted and recreated frequently by the user application. When this flag is set, Creo Parametric will not store the data on graphics card.
- *num_strips*—Specifies the number of triangle strips that will contain the triangle data.
- *strip_size*—Specifies the number of vertices in each triangle strip. It is a `ProArray` of size *num_strips*.
- *strips_array*—Specifies a `ProArray` of `ProTriVertex` structure. The structure contains information about the position and normals of the vertices for triangles.

The function `ProDispObjectAttach()` attaches the display object to a Creo Parametric scene.

After the object is attached, it will be rendered along with the Creo Parametric graphics. If the scene is regenerated, the Creo Parametric TOOLKIT application must reattach it. This API is supported only for 3D mode, that is, part, assembly, and so on and does not work in 2D mode, that is, drawings, sketch, layout, and so on. The input arguments are:

- *window*—Specifies the ID of the window in which the display object must be attached to a Creo Parametric scene.
- *obj*—Specifies the handle to the display object.
- *key_list*—Specifies a `ProArray` which contains the member identifier table of the component to which the display object will be attached. Specify `NULL` if you want to attach the display object to the top level component.
- *new_key*—Specifies the ID of the new node in the scene graph where the display object will be attached.
- *transform*—Specifies the position of the display object relative to its parent as a `ProMatrix` object.

Use the function `ProDispObjectDetach()` to detach a display object from the Creo Parametric scene. The input arguments are:

- *window*—Specifies the ID of the window from which the display object will be detached from a Creo Parametric scene.
- *key_list*—Specifies a `ProArray` which contains the member identifier table of the display object `ProDispObject` which will be detached from the scene. Pass as input *key_list* along with *new_key* defined in the API `ProDispObjectAttach()`.

The function `ProDispObjectDelete()` deletes the specified display object. You must detach the display object before deleting it.

The function `ProDispObjectSetTransform()` transforms a display object relative to its parent in the scene. Specify the new position using the `ProMatrix` object. It returns the original position of the display object as a `ProMatrix` object.

Use the function `ProDispObjectSetSurfaceAppearanceProps()` to set the display properties of the display object attached to a scene using the `ProSurfaceAppearanceProps` data structure. Refer to the section Surface Properties on page 496 for more information on `ProSurfaceAppearanceProps`.

Example 1: Creating a Display Object

The sample code in `TestDispObject.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_examples/pt_dispobj` shows how to create a display object and attach it to a scene. It also shows how to set the display properties for the display object.

21

User Interface: Selection

This chapter contains functions that enable you to select objects in Creo Parametric from within the Graphics Window or the Model Tree using the mouse or the keyboard.

The Selection Object

Like `ProModelitem`, the object `ProSelection` identifies a model item in the Creo Parametric database. `ProSelection`, however, contains more information than `ProModelitem`, and is therefore sometimes used instead of `ProModelitem` in situations where the extra information is needed. The most important use of `ProSelection` is as the output of the function for interactive selection, `ProSelect()` (thus the name ProSelection).

`ProSelection` is declared as an opaque pointer, and is, strictly speaking, a WHandle because, although the model item is a reference to a Creo Parametric database item, the other information is not.

Functions Introduced:

- **ProSelectionAlloc()**
- **ProSelectionSet()**
- **ProSelectionCopy()**
- **ProSelectionFree()**
- **ProSelectionAsmcomppathGet()**
- **ProSelectionModelitemGet()**
- **ProSelectionUvParamGet()**
- **ProSelectionViewGet()**
- **ProSelectionPoint3dGet()**
- **ProSelectionDepthGet()**
- **ProSelectionVerify()**
- **ProSelectionWindowIdGet()**
- **ProSelectionUvParamSet()**
- **ProSelectionViewSet()**
- **ProSelectionPoint3dSet()**
- **ProSelectionDrawingGet()**
- **ProSelectionDwgtblcellGet()**

Unpacking a ProSelection Object

For each item of information that `ProSelection` can contain, there is a Creo Parametric TOOLKIT function that extracts that information. The following table lists these items.

Creo Parametric TOOLKIT Function	Creo Parametric TOOLKIT Object	Meaning
`ProSelectionAsmcomppathGet()`	`ProAsmcomppath`	Assembly component path
`ProSelectionModelitemGet()`	`ProModelitem`	Model item
`ProSelectionPoint3dGet()`	`ProPoint3d`	3-D point on the model item
`ProSelectionUvParamSet()`	`ProUvParam`	`u` and `v`, or `t`, of that point
`ProSelectionDepthGet()`	double	Selection depth
`ProSelectionDistanceEval()`	double	Distance between two selected objects. The selection objects can be surfaces, entities, surface-curves, surface-nodes, or points.
`ProSelectionViewGet()`	`ProView`	Drawing view in which the selection was made
`ProSelectionWindowIdGet()`	int	Window where a selection is done
`ProSelectionDrawingGet()`	`ProDrawing`	Drawing in which the selection was made.
`ProSelectionDwgtblcellGet()`	several integers	Table segment, row, and column of a selected drawing table cell

The assembly component path is the path down from the root assembly to the model that owns the database item being referenced. It is represented by the object `ProAsmcomppath` and is described fully in the Assembly: Basic Assembly Access on page 1130 section.

The model item describes the database item in the context of its owning model, but does not refer to any parent assembly.

The 3-D point is the location, in solid coordinates, of a selected point on the model item, if it is a geometry object. The solid coordinates are those of the solid directly owning the model item.

If the model item is a surface, `ProUvParam` contains the `u` and `v` values that correspond to the 3-D selection point described above. If the item is an edge or curve, `ProUvParam` contains the `t` value.

The selection depth is the distance between the selected point and the point from which the selection search started. This is important only when you are using `ProSolidRayIntersectionCompute()`, described in the section Ray Tracing on page 194.

The view is used to distinguish different views of a solid in a drawing.

Building a ProSelection Object

Some Creo Parametric TOOLKIT function require a `ProSelection` object as an input. In many cases the assembly path—`ProAsmcomppath`—and the modelitem will be all that is needed, so `ProSelectionAlloc()` or `ProSelectionSet()` can be used. In other cases, for example when a `ProSelection` needs to identify a specific drawing view, or a specific location on a geometry item, you may also need to call functions `ProSelectionViewSet()`, `ProSelectionUvParamSet()`, and `ProSelectionPoint3dSet()`.

ProSelection Function Examples

Examples of Creo Parametric TOOLKIT functions that use `ProSelection` are as follows:

- `ProSelect()` uses `ProSelection` as its output to describe everything about the selected item.
- `ProGeomitemDistanceEval()` uses `ProSelection` as its input, instead of `ProGeomitem`, so it can measure the distance between model items in different subassemblies.
- `ProSelectionHighlight()` and `ProSelectionUnhighlight()` use `ProSelection` as inputs to distinguish different instances of the same model item in different subassemblies, and also different drawing views of the same model.
- `ProFeatureCreate()` usually uses `ProSelection` objects to identify the geometry items the feature needs to reference.
- `ProDrawingDimensionCreate()` uses `ProSelection` objects to identify the entities the dimension will attach to and the drawing view in which the dimension is to be displayed.

In a case such as `ProGeomitemDistanceEval()`, which uses `ProSelection` as an input, you might need to build a `ProSelection` object out of its component data items. The function `ProSelectionAlloc()` allocates a new `ProSelection` structure, and sets the `ProAsmcomppath` and `ProModelitem` data in it. The function `ProSelectionSet()` sets that information in a `ProSelection` object that already exists. The function `ProSelectionVerify()` checks to make sure the contents of a `ProSelection` are consistent.

The function`ProSelectionCopy()` copies one `ProSelection` object to another. `ProSelectionFree()` frees the memory of a `ProSelection` created by `ProSelectionAlloc()` or as output by a Creo Parametric TOOLKIT function.

Interactive Selection

Function Introduced:

- **ProSelect()**

`ProSelect()` is the Creo Parametric TOOLKIT function that forces the user to make an interactive graphics selection in Creo Parametric. Using this function the user can specify filters which control the items that can be selected.

Typically, the user has control over the filter options available from the `filter` menu located in the status bar at the bottom of the Creo Parametric graphics window. A call to `ProSelect()` sets application desired filters for the next expected selection.

The user interface shown when prompting for an interactive selection will be the selection dialog, with `OK` or `Cancel` buttons. Depending on the selection types permitted the user will be able to select items via:

- The graphics window
- The model tree
- The search tool
- External object
- Items external to the activated component

None of these selections or possible selections will provide guidance to the user about what to select and why. Therefore, it is important to provide detailed instructions to the user through the message window or dialog box that the application expects the user to make a selection. Use the function `ProMessageDisplay()` to explain to the user the type or purpose of the selection you want them to make.

Note

When using this function in a UI command, make sure that the command priority for the UI command is appropriate for using `ProSelect()`. Improper priority settings can cause unpredictable results. See also Normal priority actions on page in the User Interface: Menu on page 301 chapter.

The synopsis of `ProSelect()` is as follows:

```
ProError ProSelect (
  char              option[],        /* (In)  The selection filter. */
  int               max_count,       /* (In)  The maximum number of
                                                selections allowed. */
  ProSelection     *p_in_sel,        /* (In)  An array of pointers to
                                                selection structures used
                                                to initialize the array
```

```
                                                of selections. This can
                                                be NULL. */
  ProSelFunctions  *sel_func,        /* (In)  A pointer to a structure
                                                of filter functions. This
                                                can be NULL. */
  ProSelectionEnv   sel_env,         /* (In)  Use attribute PRO_SELECT_ACTIVE_
                                                COMPONENT_IGNORE to also select items
                                                external to the activate component. */
  ProSelAppAction   appl_act_data,   /* (In)  Use NULL. */
  ProSelection    **p_sel_array,     /* (Out) A pointer to an array of
                                                pointers to selected items.
                                                This argument points to static
                                                memory allocated by the
                                                function. It is reallocated
                                                on subsequent calls to this
                                                function.*/
  int              *p_n_sels         /* (Out) The actual number of
                                                selections made. The
                                                function allocates the
                                                memory for this function
                                                and reuses it on
                                                subsequent calls.*/
)
```

The first input argument to `ProSelect()`, *option*, is the set of item types that can be selected. This is in the form of a C string which contains the names of the types separated by commas (but no spaces). The following table lists the item types that can be selected by `ProSelect()`, the name of the type that must appear in the *option* argument, and the value of `ProType` contained in the `ProModelitem` for the selected item.

Creo Parametric Database Item	ProSelect() Option	ProType
Geometry Items		
Datum point	point	`PRO_POINT`
Datum axis	axis	`PRO_AXIS`
Datum plane	datum	`PRO_SURFACE`
Coordinate system datum	csys	`PRO_CSYS`
Coordinate System Axis	csys_axis	`PRO_CSYS_AXIS_X` `PRO_CSYS_AXIS_Y` `PRO_CSYS_AXIS_Z`
Edge (solid or datum surface)	edge	`PRO_EDGE`
Vertex	edge_end	`PRO_EDGE_START`, or `PRO_EDGE_END`
Datum curve	curve	`PRO_CURVE`
Datum curve end	curve_end	`PRO_CRV_START`, or `PRO_CRV_END`
Composite Curve	comp_crv	`PRO_CURVE`
Edge (solid only)	sldedge	`PRO_EDGE`
Edge (datum surface only)	qltedge	`PRO_EDGE`

Creo Parametric Database Item	ProSelect() Option	ProType
Pipe segment end	pipeseg_end	PRO_PSEG_START or PRO_PSEG_END
Surface (solid or quilt)	surface	PRO_SURFACE
Surface (solid)	sldface	PRO_SURFACE
Surface (datum surface)	qltface	PRO_SURFACE
Surface (point)	pntsrf	PRO_SURFACE_PNT
Quilt	dtmqlt	PRO_QUILT
Annotations		
Dimension	dimension	PRO_DIMENSION
Reference dimension	ref_dim	PRO_REF_DIMENSION
Geometric tolerance	gtol	PRO_GTOL
3D symbol	symbol_3d	PRO_SYMBOL_INSTANCE
Note	any_note	PRO_NOTE
3D Note	note_3d	PRO_NOTE
3D surface finish	surffin_3d	PRO_SURF_FIN
Annotation element	annot_elem	PRO_ANNOTATION_ELEM
Drawing Items		
Drawing view	dwg_view	PRO_VIEW
Drawing table	dwg_table	PRO_DRAW_TABLE
Draft entity	draft_ent	PRO_DRAFT_ENTITY
Detail symbol	dtl_symbol	PRO_SYMBOL_INSTANCE
Detail note	any_note	PRO_NOTE
Table cell	table_cell	PRO_DRAW_TABLE
Solids and Features		
Feature	feature	PRO_FEATURE
Part	part	PRO_PART
Component Feature	membfeat	PRO_FEATURE
Assembly component model	component	PRO_PART, PRO_ASSEMBLY
Part or subassembly	prt_or_asm	PRO_PART, or PRO_ASSEMBLY
Miscellaneous Items		
Creo Simulate Items	Refer to the table in the section Selection of Creo Simulate Items on page 1855	
External object	ext_obj	PRO_EXTOBJ
Diagram fixed connector, fixed component, or parametric connector	dgm_obj	PRO_DIAGRAM_OBJECT
Diagram wire (not a cable)	dgm_non_cable_wire	PRO_DIAGRAM_OBJECT
Solid Body	3d_body	PRO_BODY
ECAD conductor	ecad_cu	PRO_ECAD_CONDUCTOR

The second argument specifies the maximum number of items the user can select. If there is no maximum, set this argument to -1.

The third argument to `ProSelect()` is an expandable array of `ProSelection` structures (created using `ProArrayAlloc()` and `ProSelectionAlloc()`) used to initialize the selection list. For more information refer to section Expandable Arrays on page 59 in the chapter Fundamentals on page 22. This is used in situations like **Feature**, **Define** in Creo Parametric where the user has the option of removing a default selection for a feature reference.

The fourth argument is an optional structure that specifies three, user-defined filter functions. These enable you to filter the items that are selectable in a customized way. For example, you could arrange that only straight edges are selectable by writing a filter that would check the type of the edge, and return an appropriate status. This function would then be called within `ProSelect()` to prevent the user from selecting a curved edge.

The fifth argument allows the user to pass a set of attributes to `ProSelect()` using the function `ProSelectionEnvAlloc()`. The function `ProSelectionEnvAlloc()` returns the `ProSelectionEnv` handle which is given as input to `ProSelect()`. The attributes of `ProSelectionEnvAlloc()` are:

- `PRO_SELECT_DONE_REQUIRED`—Specifies that user has to click **OK** in the **Select** dialog box to get the selected items.
- `PRO_SELECT_BY_MENU_ALLOWED`—Specifies that search tool is available in the function `ProSelect()` when the attribute value is set to True, which is the default value.
- `PRO_SELECT_BY_BOX_ALLOWED`—Specifies that user must draw a bounding box to get the items selected within the box.

> **Note**
>
> The attribute `PRO_SELECT_BY_BOX_ALLOWED` can be used only for the types specified under “Geometry Items” in the above table.

- `PRO_SELECT_ACTIVE_COMPONENT_IGNORE`—Specifies that user can select items external to the activate component.
- `PRO_SELECT_HIDE_SEL_DLG`—Specifies that the **Select** dialog box must be hidden.

The sixth argument is not used in this release and should be set to `NULL`.

The final two arguments are the outputs, that is, an expandable array of `ProSelection` structures, and the number of items in the array. The previous section explains how to analyze the contents of a `ProSelection` object.

Note

The array of ProSelections is allocated and reused on subsequent calls to `ProSelect()`.

Therefore, you must not free the output array. Also, if you wish to preserve any of the selections made, you should copy that selection using `ProSelectionCopy()`.

Note

- When using the function `ProSelect()` from within a loop, if you encounter the error `PRO_TK_PICK_ABOVE (-14)`, then you must handle this error by returning control back to Creo Parametric.
- The function `ProSelect()` returns `PRO_TK_NO_ERROR` when you end the command without making any selection in Creo Parametric. The function returns the output argument `p_n_sels` as zero and the array `p_sel_array` as `NULL`.

Highlighting

Functions Introduced:

- **ProSelectionHighlight()**
- **ProSelectionDisplay()**
- **ProSelectionUnhighlight()**

The function `ProSelectionHighlight()` highlights an item specified by a `ProSelection` object in a color chosen from the enumerated type `ProColortype`. This highlight is the same as the one used by Creo Parametric (and `ProSelect()`) when selecting an item—it just repaints the wire-frame display in the new color. The highlight is removed if you use the View Repaint command or `ProWindowRepaint()`; it is not removed if you use `ProWindowRefresh()`.

The function `ProSelectionUnhighlight()` removes the highlight.

`ProSelectionHighlight()` will not change the highlight color of an item already highlighted. If you need to do this, call `ProSelectionUnhighlight()` on the first item.

The function `ProSelectionDisplay()` does the same highlight as `ProSelectionHighlight()` but uses the standard highlight color used by Creo Parametric.

Note

For performance reasons, calls to `ProSelectionHighlight()` are cached and executed after a short delay or after a window repaint.

Selection Buffer

Introduction to Selection Buffers

Selection is the process of choosing items on which you want to perform an operation. In Creo Parametric, before a feature tool is invoked, the user can select items to be used in a given tool's collectors. Collectors are like storage bins of the references of selected items. The location where preselected items are stored is called the selection buffer.

Depending on the situation, different selection buffers may be active at any one time. In Part and Assembly mode, Creo Parametric offers the default selection buffer, the Edit selection buffer, and other more specialized buffers. Other Creo Parametric modes offer different selection buffers.

In the default Part and Assembly buffer there are two levels at which selection is done:

- First Level Selection

Provides access to higher-level objects such as features or components. You can make a second level selection only after you select the higher-level object.

- Second Level Selection

Provides access to geometric objects such as edges and faces.

Note

First-level and second-level objects are usually incompatible in the selection buffer.

Creo Parametric TOOLKIT allows access to the contents of the currently active selection buffer. The available functions allow your application to:

- Get the contents of the active selection buffer.
- Remove the contents of the active selection buffer.
- Add to the contents of the active selection buffer.

Reading the Contents of the Selection Buffer

Functions Introduced:

- **ProSelbufferSelectionsGet()**
- **ProSelectionCollectionGet()**

Use the function `ProSelbufferSelectionsGet()` to access the contents of the current selection buffer. The function returns independent copies of the selections in the selection buffer (if the buffer is cleared, this array is still valid).

If there is no active selection buffer this function returns an error.

Use the standard Creo Parametric TOOLKIT functions to parse the contents of the `ProSelection` array.

However, the selection buffer stores chain and surface collections using a special mechanism. The function `ProSelectionCollectionGet()` can be used to extract the `ProCollection` object from a `ProSelection`.

Note

As per the manner of storage of the collection in the selection buffer, once the collection has been cleared from the buffer the `ProSelection` referring to the collection is no longer valid. Therefore it is recommended to extract the `ProCollection` object from the `ProSelection` before there is a possibility that it may be cleared from the selection buffer.

Removing the Items from the Selection Buffer

Functions Introduced:

- **ProSelbufferClear()**
- **ProSelbufferSelectionRemove()**

Use the function `ProSelbufferClear()` to clear the currently active selection buffer of all contents. After the buffer is cleared, all contents are lost.

Use the function `ProSelbufferSelectionRemove()` to remove a specific selection from the selection buffer. The input argument is the index of the item (the index where the item was found in the call to `ProSelbufferSelectionsGet()`).

Note

Because of the specialized nature of the Edit buffer in Creo Parametric, modification of the contents of the Edit buffer is not supported.

Adding Items to the Selection Buffer

Functions Introduced:

- **ProSelbufferSelectionAdd()**
- **ProSelbufferCollectionAdd()**

Use the function `ProSelbufferSelectionAdd()` to add an item to the active selection buffer.

Note

The selected item must refer to an item that is in the current model such as its owner, component path or drawing view.

This function may fail due to any of the following reasons:

- There is no current selection buffer active.
- The selection does not refer to the current model.
- The item is not currently displayed and so cannot be added to the buffer.
- The selection cannot be added to the buffer in combination with one or more objects that are already in the buffer. For example: geometry and features cannot be selected in the default buffer at the same time.

Use the function `ProSelbufferCollectionAdd()` to add a chain or surface collection to the active selection buffer. It has similar restrictions as `ProSelbufferSelectionAdd()`.

Note

because of the specialized nature of the Edit buffer in Creo Parametric, modification of the contents of the Edit buffer is not supported.

Refer to the sample code Example 3: Assigning Creo Parametric command to popup menus on page 320. This example demonstrates the use of the selection buffer to determine the context for a popup menu button.

22

User Interface: Curve and Surface Collection

This chapter describes the Creo Parametric TOOLKIT functions to access the details of curve and surface collections for query and modification. Curve and surface collections are required inputs to a variety of Creo Parametric tools such as Round, Chamfer, and Draft.

Introduction to Curve and Surface Collection

A curve collection or chain is a group of separate edges or curves that are related, for example, by a common vertex, or tangency. Once selected, these separate entities are identified as a chain so they can be modified as one unit.

The different chain types are as follows:

- One-by-one—a chain of edges, curves or composite curves, each adjacent pair of which has a coincident endpoint. Some applications may place other conditions on the resulting chain.
- Tangent—a chain defined by the selected item and the extent to which adjacent entities are tangent.
- Curve—an entire composite curve or some portion of it that is defined by two component curves of the curve.
- Boundary—an entire loop of one-sided edges that bound a quilt or some portion thereof defined by two edges of the boundary loop.
- Surf Chain—an entire loop of edges that bound a face (solid or surface) or some portion of it that is defined by two edges of the loop.
- Intent Chain—an intent chain entity, usually created as the result of two intersecting features.
- From/To—a chain that begins at a start-point, follows an edge line, and ends at the end-point.

Surface sets are one or more sets of surfaces either for use within a tool, or before entering a tool.

The definition of a surface set may not be independent in all respects from that of any other. In other words, the ability to construct some types of surface sets may depend on the presence of or on the content of others. On this account, we have different surface sets as follows:

- One-by-One Surface Set—Represents a single or a set of single selected surfaces, which belong to solid or surface geometry.
- Intent Surface Set—Represents a single or set of intent surfaces, which are used for the construction of the geometry of features. This instruction facilitates the reuse of the feature construction surface geometry as "intent" reference. This is also known as "logical object surface set".
- All Solid or Quilt Surface Set—Represents all the solid or quilt surfaces in the model.

- Loop Surface Set—Represents all the surfaces in the loop in relation with the selected surface and the edge. This is also known as "neighboring surface set".
- Seed and Boundary Surface Set—Represents all the surfaces between the seed surface and the boundary surface, excluding the boundary surface itself.

Surface set collection can be identified as a gathering a parametric set of surfaces in the context of a tool that specifically requests surface sets and is nearly identical to selection of a surface set.

Chain collection can be identified as a gathering a chain in the context of a tool that specifically requests chain objects and is nearly identical to chain selection.

Collection is related to Selection as follows:

Selection is the default method of interaction with Creo Parametric system. Selection is performed without the context of any tool. In other words, the system does not know what to do with selected items until the user tells the system what to do. Collection is essentially Selection within the context of a tool. Items are gathered for a specific use or purpose as defined by the tool, which forms the Collection. It is possible to convert the collections into the sets of selections using the collection APIs.

The ProCollection object

A `ProCollection` object is an opaque pointer to a structure in which a surface or curve collection is stored. It represents a chain or surface set and extracts the details from an appropriate structure from Creo Parametric.

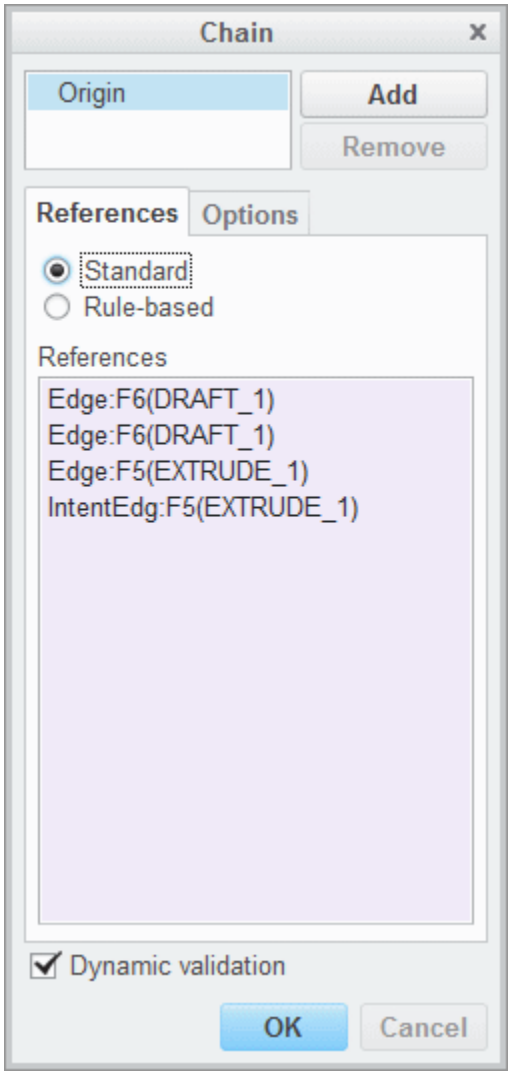
Interactive Collection

Functions Introduced:

- **ProCurvesCollect()**
- **ProSurfacesCollect()**

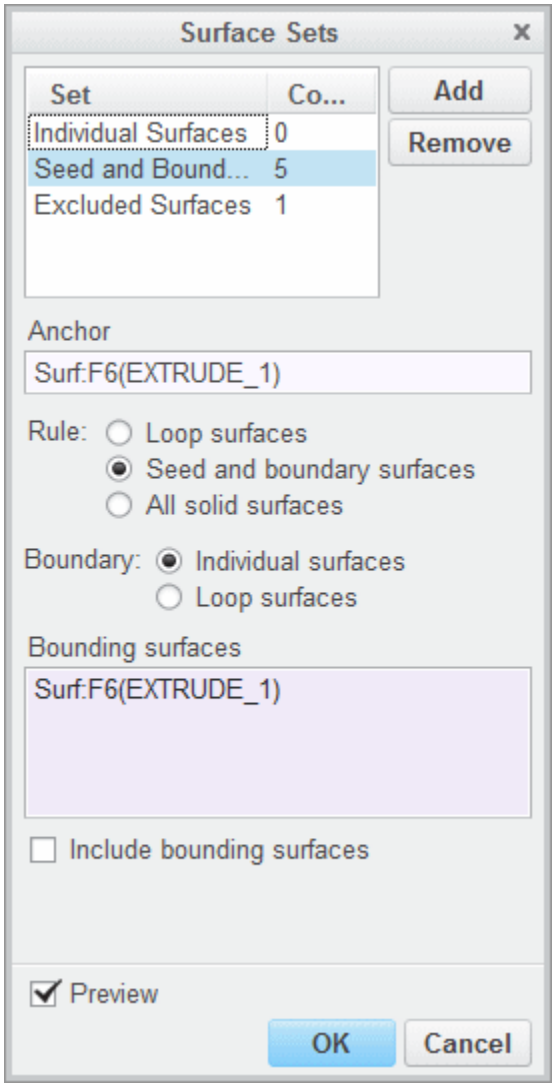
Use the function `ProCurvesCollect()` to interactively create a collection of curves by invoking a chain collection user interface.

Chain collection User Interface



Use the function `ProSurfacesCollect()` to interactively create a collection of surfaces by invoking a surface sets dialog.

Surface Sets dialog box



For the functions `ProCurvesCollect()` and `ProSurfacesCollect()` the input arguments are as follows:

- *types*—Specifies an array defining the permitted instruction types. The following instruction types are supported:
  - `PRO_CHAINCOLLUI_ONE_BY_ONE`—for creating "One by One" chain.
  - `PRO_CHAINCOLLUI_TAN_CHAIN`—for creating "Tangent" chain.
  - `PRO_CHAINCOLLUI_CURVE_CHAIN`—for creating "Curve" chain.
  - `PRO_CHAINCOLLUI_BNDRY_CHAIN`—for creating "Boundary Loop" chain.
  - `PRO_CHAINCOLLUI_FROM_TO`—for creating "From-To" chain.

- PRO_CHAINCOLLUI_ALLOW_LENGTH_ADJUSTMENT—for allowing length adjustment in curves.
- PRO_CHAINCOLLUI_ALLOW_ALL—for allowing all the supported instruction types.
- PRO_CHAINCOLLUI_ALLOW_EXCLUDED—for excluding chain.
- PRO_CHAINCOLLUI_ALLOW_APPENDED—for appending chain.

- *n_types*—Specifies the size of the types array.
- *filter_func*—Specifies the filter function. The filter function is called before each selection is accepted. If your application wishes to reject a certain selection, it can return an error status from this filter. You may pass NULL to skip the filter.
- *app_data*—Specifies the application data that will be passed to filter_func. You may pass NULL when this is not required.
- *collection*—Specifies the collection object where the results will be stored. This should be preallocated using ProCollectionAlloc() (for ProCurvesCollect()) and ProSrfcollectionAlloc() (for ProSurfacesCollect()) respectively. The collection properties and instructions will be stored in this handle after the call to the functions ProCurvesCollect() and ProSurfacesCollect(). Use the functions for programmatic access to curve and surface collections and to extract the required information.

The output arguments are as follows:

- *sel_list*—Specifies a pointer to an array of ProSelection objects describing the current resulting curves and edges or surfaces resulting from the collection. You may pass NULL if you are not currently interested in the results of the collection.
- *n_sel*—Specifies the number of entries in sel_list.

Accessing Collection object from Selection Buffer

Function Introduced:

- **ProSelectionCollectionGet()**

The selection buffer stores chain and surface collections using a special mechanism. The function ProSelectionCollectionGet() can be used to extract the ProCollection object from a ProSelection. The function ProSelectionCollectionGet() may fail due to the following conditions:

- PRO_TK_INVALID_TYPE—The selection object does not contain a collection.
- PRO_TK_INVALID_PTR—The selection object contains a chain, but this chain reference is no longer valid. Chain references contained in ProSelections are only valid for as long as the chain is selected in the selection buffer.

Note

It is recommended to extract the ProCollection object from the ProSelection before it can be cleared from the selection buffer.

Adding a Collection Object to the Selection Buffer

Function Introduced:

- **ProSelbufferCollectionAdd()**

Use the function ProSelbufferCollectionAdd() to add a chain or surface collection to the active selection buffer. The function ProSelbufferCollectionAdd() may fail due to the following conditions:

- PRO_TK_INVALID_ITEM—The collection does not correctly reference the current model.
- PRO_TK_NOT_DISPLAYED—The collection contains one or more items that are not currently displayed (for example, due to inactive geometry) and so cannot be added to the buffer.
- PRO_TK_BAD_CONTEXT—The selection cannot be added to the buffer in combination with one or more objects that are already in the buffer (for example geometry and features selected at the same time).

For further information on related functions, refer to the section Selection Buffer on page 512 of chapter User Interface: Selection on page 503.

Programmatic Access to Collections

A ProCollection object may be returned from some indeterminate contexts or functions, however it can only be one of the following types:

- PRO_COLLECTION_LEGACY—legacy curve collection type (generated by ProCurvesCollect() or a superseded function like ProCollectionAlloc().
- PRO_COLLECTION_SRFCOLL—surface collection type.
- PRO_COLLECTION_CRVCOLL—curve collection type.

Function Introduced:

- **ProCollectionTypeGet()**

Use the function ProCollectionTypeGet()ProCollectionType

Contents of Curve Collection

Functions Introduced:

- **ProCrvcollectionInstructionsGet()**
- **ProCrvcollinstrTypeGet()**
- **ProCrvcollinstrAttributeIsSet()**
- **ProCrvcollinstrReferencesGet()**
- **ProCrvcollinstrValueGet()**
- **ProCrvcollinstrFree()**
- **ProCrvcollinstrArrayFree()**

Use the function ProCrvcollectionInstructionsGet() to get the instructions from the curve collection.

Use the function ProCrvcollinstrTypeGet() to get the curve collection instruction type.

Curve collection instructions can be of the following types:

- PRO_CURVCOLL_EMPTY_INSTR—to be used when you do not want to pass any other instruction.
- PRO_CURVCOLL_ADD_ONE_INSTR—for creating "One by One" chain.
- PRO_CURVCOLL_TAN_INSTR—for creating "Tangent" chain.
- PRO_CURVCOLL_CURVE_INSTR—for creating "Curve" chain.
- PRO_CURVCOLL_SURF_INSTR—for creating "Surface Loop" chain.
- PRO_CURVCOLL_BNDRY_INSTR—for creating "Boundary Loop" chain.
- PRO_CURVCOLL_LOG_OBJ_INSTR—for creating "Logical Object" chain.
- PRO_CURVCOLL_PART_INSTR—for creating chain on all possible references, or to choose from convex or concave only.

- PRO_CURVCOLL_FEATURE_INSTR—for creating chain from feature curves.
- PRO_CURVCOLL_FROM_TO_INSTR—for creating "From-To" chain.
- PRO_CURVCOLL_EXCLUDE_ONE_INSTR—for excluding the entity from the chain.
- PRO_CURVCOLL_TRIM_INSTR—to trim chain.
- PRO_CURVCOLL_EXTEND_INSTR—to extend chain.
- PRO_CURVCOLL_START_PNT_INSTR—to set the chain start point.
- PRO_CURVCOLL_ADD_TANGENT_INSTR—to add all edges tangent to the ends of the chain.
- PRO_CURVCOLL_ADD_POINT_INSTR—to add selected point or points to the collection.
- PRO_CURVCOLL_OPEN_CLOSE_LOOP_INSTR—to add a closed chain that is considered as open.
- PRO_CURVCOLL_QUERY_INSTR—for creating “Query” chain.
- PRO_CURVCOLL_CNTR_INSTR—to add contours to the collection.
- PRO_CURVCOLL_SRFS_BNDRY_INSTR—to collect boundary of the given set of surfaces.
- PRO_CURVCOLL_SRFS_BNDRY_ADJ_INSTR—to collect edges adjacent to the boundary of the given surfaces.
- PRO_CURVCOLL_SKET_ADD_ONE_INSTR—for creating “One by One” in sketcher.
- PRO_CURVCOLL_SKET_FROM_TO_INSTR—for creating “From-to” sketcher chain.
- PRO_CURVCOLL_RESERVED_INSTR—to determine the number of instructions defined in the curve instruction.

Use the function ProCrvcollinstrAttributeIsSet() to check whether a special attribute is set for the curve collection instruction.

The curve collection instruction attributes can be of the following types:

- PRO_CURVCOLL_NO_ATTR—applicable when there are no attributes present.
- PRO_CURVCOLL_ALL—applicable for all edges.
- PRO_CURVCOLL_CONVEX—applicable for convex edges only.
- PRO_CURVCOLL_CONCAVE—applicable for concave edges only.

Use the function ProCrvcollinstrReferencesGet() to get the references contained in a curve collection instruction.

Use the function ProCrvcollinstrValueGet() to get the value of a curve collection instruction. This is valid for instructions of PRO_CURVCOLL_TRIM_INSTR and PRO_CURVCOLL_EXTEND_INSTR type.

Use the function ProCrvcollinstrFree() to release a curve collection instruction.

Use the function ProCrvcollinstrArrayFree() to release the ProArray of curve collection instructions.

Creation and Modification of Curve Collections

Functions Introduced:

- **ProCrvcollectionAlloc()**
- **ProCrvcollectionCopy()**
- **ProCrvcollectionInstructionAdd()**
- **ProCrvcollectionInstructionRemove()**
- **ProCrvcollinstrAlloc()**
- **ProCrvcollinstrAttributeSet()**
- **ProCrvcollinstrAttributeUnset()**
- **ProCrvcollinstrReferenceAdd()**
- **ProCrvcollinstrReferenceRemove()**
- **ProCrvcollinstrValueSet()**
- **ProCrvcollectionRegenerate()**

Use the function ProCrvcollectionAlloc() to allocate a curve collection.

Use the function ProCrvcollectionCopy() to copy a curve collection which is a newly allocated collection object internally and can be freed from memory using the function ProCollectionFree().

Note

ProCrvcollectionCopy() function should be used to convert the collection object returned by ProCurvesCollect() to a PRO_COLLECTION_CRVCOLL type of collection so that it can be used by the curve collection access functions.

Use the function ProCrvcollectionInstructionAdd() to add an instruction to a curve collection.

Use the function ProCrvcollectionInstructionRemove() to remove an instruction from a curve collection.

Use the function `ProCrvcollinstrAlloc()` to allocate a curve collection instruction.

Use the function `CrvcollinstrAttributeSet()` to add an attribute to the curve collection instruction.

Use the function `ProCrvcollinstrAttributeUnset()` to remove an attribute of a curve collection instruction.

Use the function `ProCrvcollinstrReferenceAdd()` to add a reference to curve collection instruction references.

Use the function `ProCrvcollinstrReferenceRemove()` to remove a reference to curve collection instruction.

Use the function `ProCrvcollinstrValueSet()` to set value of the curve collection instruction.

Use the function `ProCrvcollectionRegenerate()` to generate an array of objects based on the rules and information in the collection.

Contents of Surface Collection

Functions Introduced:

- **ProSrfcollectionInstructionsGet()**
- **ProSrfcollinstrTypeGet()**
- **ProSrfcollinstrIncludeGet()**
- **ProSrfCollinstrInfoGet()**
- **ProSrfcollinstrReferencesGet()**
- **ProSrfcollinstrArrayFree()**
- **ProSrfcollinstrFree()**
- **ProSrfcollrefTypeGet()**
- **ProSrfcollrefItemGet()**
- **ProSrfcollrefArrayFree()**
- **ProSrfcollrefFree()**

Use the function `ProSrfcollectionInstructionsGet()` to get an array of instructions assigned to the surface collection.

Use the function `ProSrfcollinstrTypeGet()` to get the type of surface collection instruction.

Surface collection instructions can be of the following types:

- PRO_SURFCOLL_SINGLE_SURF—Instruction specifying a set of single surfaces.
- PRO_SURFCOLL_SEED_N_BND—Instruction specifying a combination of Seed and Boundary type of surfaces.
- PRO_SURFCOLL_SEED_N_BND_INC_BND—Instruction specifying a combination of Seed and Boundary type of surfaces and also includes the seed surfaces.
- PRO_SURFCOLL_QUILT_SRFS—Instruction specifying quilt type of surfaces.
- PRO_SURFCOLL_ALL_SOLID_SRFS—Instruction specifying all solid surfaces in the model.
- PRO_SURFCOLL_NEIGHBOR—Instruction specifying neighbor type of surfaces (boundary loop).
- PRO_SURFCOLL_NEIGHBOR_INC—Instruction specifying neighbor type of surfaces (boundary loop) and also includes the seed surfaces.
- PRO_SURFCOLL_LOGOBJ_SRFS—Instruction specifying intent surfaces. Intent surfaces are also known as "logical objects".
- PRO_SURFCOLL_GEOM_RULE—— Instruction specifying collection of surfaces using geometry rules.
- PRO_SURFCOLL_SHAPE_BASED—Instruction specifying collection of shape based surfaces.
- PRO_SURFCOLL_TANG_SRF—Instruction specifying collection of tangent surfaces.

The following flags are used as an input to ProSurfacesCollect and drive its behavior of the interactive collection.

- PRO_SURFCOLL_DISALLOW_QLT—Do not allow selections from quilts.
- PRO_SURFCOLL_DISALLOW_SLD—Do not allow selections from solid geometry.
- PRO_SURFCOLL_DONT_MIX—Allow selections from only solid or only quilt but no mixing.
- PRO_SURFCOLL_SAME_SRF_LST—Allow selections from same solid or same quilt.
- PRO_SURFCOLL_USE_BACKUP—Prompts Creo Parametric to regenerate using backups.
- PRO_SURFCOLL_DONT_BACKUP—Do not back up copy of references.

- PRO_SURFCOLL_DISALLOW_LOBJ—Do not allow selections from intent surfaces or logical objects.
- PRO_SURFCOLL_ALLOW_DTM_PLN—Allows datum plane selection. It is not supported in Pro/ENGINEER Wildfire 2.0.

Use the function ProSrfcollinstrIncludeGet() to check whether the include flag of the surface collection instruction is set. If the include flag is PRO_B_TRUE, the surfaces generated by this instruction add surfaces to the overall set. If the include flag is PRO_B_FALSE, the surfaces generated by this instruction are removed from in the overall set.

Use the function ProSrfCollinstrInfoGet() to get the information about the bit flags from the surface collection instruction. For more information on bit flags, see the function ProSrfCollinstrInfoSet() in the section Creation and Modification of Surface Collections on page 528.

Use the function ProSrfcollinstrReferencesGet() to get the references contained in a surface collection instruction.

Use the function ProSrfcollinstrArrayFree() to free a ProArray of surface collection reference handles.

Use the function ProSrfcollinstrFree() to release the surface collection instructions.

Use the function ProSrfcollrefTypeGet() to get the type of reference contained in the surface collection reference.

Surface collection references can be of the following types:

- PRO_SURFCOLL_REF_SINGLE—Specifying the collection reference belonging to the "single surface set" type of instruction. This type of reference can belong to single surface type of instruction.
- PRO_SURFCOLL_REF_SINGLE_EDGE—Specifying the collection reference belonging to the an "single surface set" edge type of instruction.
- PRO_SURFCOLL_REF_SEED—Specifying the collection reference to be the seed surface. This type of reference can belong to seed and boundary type of instruction.
- PRO_SURFCOLL_REF_SEED_EDGE—Specifying the collection reference of seed edge type. This type of reference can belong to seed and boundary type of instruction.
- PRO_SURFCOLL_REF_BND—Specifying the collection reference to be a boundary surface. This type of reference can belong to seed and boundary type of instruction.

  A single seed and boundary type of instruction will have at least one of each seed and boundary type of reference.

- PRO_SURFCOLL_REF_NEIGHBOR—Specifying the collection reference to be of neighbor type. This type of reference belongs neighbor type of instruction.
- PRO_SURFCOLL_REF_NEIGHBOR_EDGE—Specifying the collection reference of neighbor edge type. This type of reference belongs to neighbor type of instruction.

  A neighbor type of instruction will have one neighbor and one neighbor edge type of reference.
- PRO_SURFCOLL_REF_GENERIC—Specifying the collection reference to be of generic type. This type of reference can belong to intent surfaces , quilt and all-solid type of instructions.

Use the function ProSrfcollrefArrayFree() to free a ProArray of surface collection reference handles.

Use the function ProSrfcollrefFree() to free a surface collection reference handle.

Use the function ProSrfcollrefItemGet() to get the geometry item contained in the surface collection reference.

Creation and Modification of Surface Collections

Functions Introduced:

- **ProSrfcollectionAlloc()**
- **ProSrfcollectionCopy()**
- **ProSrfcollectionInstructionAdd()**
- **ProSrfcollectionInstructionRemove()**
- **ProSrfcollinstrAlloc()**
- **ProSrfcollinstrIncludeSet()**
- **ProSrfCollinstrInfoSet()**
- **ProSrfcollinstrReferenceAdd()**
- **ProSrfcollinstrReferenceRemove()**
- **ProSrfcollrefAlloc()**
- **ProSrfcollectionRegenerate()**

Use the function ProSrfcollectionAlloc() to allocate a surface collection.

Use the function ProSrfcollectionCopy() to copy a surface collection into a newly allocated ProCollection handle.

Use the function `ProSrfcollectionInstructionAdd()` to add an instruction to surface collection.

Use the function `ProSrfcollectionInstructionRemove()` to remove the instructions from surface collections.

The function `ProSrfcollectionInstructionRemove()` may fail due to invalid index specifications for the instruction or if the collection contains instruction for curves.

Use the function `ProSrfcollinstrAlloc()` to allocate a surface collection instruction.

Use the function `ProSrfcollinstrIncludeSet()` to set the `include` flag for a surface collection instruction.

Note

In the functions `ProSrfcollinstrAlloc()` and `ProSrfcollinstrIncludeSet()` to exclude the surfaces generated in the collection instruction pass the input arguments as follows:

- Specify the surface collection instruction as `PRO_SURFCOLL_SINGLE_SURF`.
- Specify the *include* argument as `PRO_B_FALSE`.

Use the function `ProSrfCollinstrInfoSet()` to set the information about the bit flags in the surface collection instruction. You can pass the bit maps information as bitmask containing one or more bit flags.

The following bit flags can be passed as input argument:

- `PRO_SURFCOLL_ALL_GEOM_RULE`—Specifies that when this flag is included with other rule flags, all the rules are applied. Otherwise, any applicable geometry rule is applied.
- `PRO_SURFCOLL_CO_PLANNAR_GEOM_RULE`—Specifies that the surfaces coplanar to the seed surface should be collected.
- `PRO_SURFCOLL_PARALLEL_GEOM_RULE`—Specifies that the surfaces parallel to the seed surface should be collected.
- `PPRO_SURFCOLL_CO_AXIAL_GEOM_RULE`—Specifies that the surfaces coaxial with the seed surface should be collected.
- `PRO_SURFCOLL_EQ_RADIUS_GEOM_RULE`—Specifies that the surfaces with the same radius and type as the seed surface should be collected.
- `PRO_SURFCOLL_SAME_CONVEXITY_GEOM_RULE`—Specifies that the surfaces that have the same convexity and type as the seed surface should be collected.

Note

The bit flags *PRO_SURFCOLL_ALL_GEOM_RULE*, PRO_SURFCOLL_CO_PLANNAR_GEOM_RULE, PRO_SURFCOLL_PARALLEL_GEOM_RULE, PRO_SURFCOLL_CO_AXIAL_GEOM_RULE, PRO_SURFCOLL_EQ_RADIUS_GEOM_RULE, and PRO_SURFCOLL_SAME_CONVEXITY_GEOM_RULE are related to PRO_SURFCOLL_GEOM_RULE, that is, the geometry rule surface set.

- PRO_SURFCOLL_SHAPE_CHAMFER—Collects surfaces with chamfered edges. If used alone then only primary chamfer shapes are added to the collection set.
- PRO_SURFCOLL_SHAPE_ROUND—Collects surfaces with round shape. If used alone then only the primary round shapes are added to the collection set.
- PRO_SURFCOLL_SHAPE_PROTR_BOSS—Collects surfaces of the type boss. The protrusion surfaces with the secondary shapes are also added to the set.
- PRO_SURFCOLL_SHAPE_PROTR_RIB—Collects surfaces of the type rib. The protrusion surfaces without the secondary shapes are added to set.
- PRO_SURFCOLL_SHAPE_CUT_POCKET—Collects cut surfaces with the secondary shapes.
- PRO_SURFCOLL_SHAPE_CUT_SLOT—Collects cut surfaces without the secondary shapes being added to set.
- PRO_SURFCOLL_SHAPE_MORE_SHAPES—Use this flag only with PRO_SURFCOLL_SHAPE_CHAMFER and PRO_SURFCOLL_SHAPE_ROUND to add the secondary chamfer and round shapes to the surface set.

Note

The bit flags PRO_SURFCOLL_SHAPE_ROUND, PRO_SURFCOLL_SHAPE_PROTR_BOSS, PRO_SURFCOLL_SHAPE_PROTR_RIB, PRO_SURFCOLL_SHAPE_CUT_POCKET, PRO_SURFCOLL_SHAPE_CUT_SLOT, PRO_SURFCOLL_SHAPE_MORE_SHAPES and PRO_SURFCOLL_SHAPE_CHAMFER are related to *PRO_SURFCOLL_SHAPE_BASED*, that is, the shape surface set.

- PRO_SURFCOLL_TANGENT_NEIGBOURS_ONLY—Specifies that the surfaces that have at least one tangent edge with another surface in the tangent surface set, starting from the seed surface should be collected.

Note

The bit flag `PRO_SURFCOLL_TANGENT_NEIGBOURS_ONLY` is related to `PRO_SURFCOLL_TANG_SRF`, that is, the tangent surface set.

Note

- Before you call `ProSrfCollinstrInfoSet()` function, you must set the include flag (`PRO_B_TRUE` / `PRO_B_FALSE`) for the surface collection instruction using the `ProSrfcollinstrIncludeSet()`.
- For the geometry based, the shape based, and the tangent based surface collections, the surface collection reference type (first input argument to the function `ProSrfcollrefAlloc()`) must be `PRO_SURFCOLL_REF_SEED`.
- If you want to apply all the geometry rule flags included in the info argument of the function `ProSrfCollinstrInfoSet()`, the flag `PRO_SURFCOLL_ALL_GEOM_RULE` must be included in info argument. If you do not include the `PRO_SURFCOLL_ALL_GEOM_RULE` flag, any of the geometry rule flags will be applied.

Use the function `ProSrfcollinstrReferenceAdd()` to add a reference to the surface collection instruction.

Use the function `ProSrfcollinstrReferenceRemove()` to remove a reference from the surface collection instruction.

Use the function `ProSrfcollectionRegenerate()` to generate an array of objects based on the rules and information in the collection.

Access of Collection Object from Feature Element Trees

Functions Introduced:

- **ProElementCollectionGet()**
- **ProElementCollectionSet()**
- **ProElementCollectionProcess()**

Use the function `ProElementCollectionGet()` to extract a collection object from an element of a feature element tree of the following types:

- PRO_E_STD_CURVE_COLLECTION_APPL
- PRO_E_STD_SURF_COLLECTION_APPL

Use the function ProElementCollectionSet() to assign a collection object into the appropriate elements of the above mentioned types.

Note

Point collections (PRO_E_POINT_COLLECTION_APPL) are accessed from the element as normal ProReference object—by using ProElementReferenceGet() and ProElementReferenceSet() APIs.

Use the function ProElementCollectionProcess() to generate a list of geometric items in the form of array of ProReference object constituting the collection. It returns a ProArray of the selected curves, edges, or surfaces that exist in the collection owned by the element. This function differs from ProCrvcollectionRegenerate() and ProSrfcollectionRegenerate() because those functions generate a list of collected entities based on standard rules. The function ProElementCollectionProcess(), if applied to a collection element from a given feature, will use the feature's rules to process the collection and return the exact geometric entities used by the feature.

Note

If this element is extracted from an existing feature tree using ProFeatureElemtreeExtract(), the returned reference_array will be on the basis of the feature rules. On the other hand, if this element is newly created and not yet assigned to a feature, then the returned reference_array will be as per the default rules.

Programmatic Access to Legacy Collections

Functions Introduced:

- **ProCollectionAlloc()**
- **ProCollectionrefAlloc()**
- **ProCollectionrefSelectionSet()**

- **ProCollectionrefTypeSet()**
- **ProCollectioninstrAlloc()**
- **ProCollectioninstrTypeSet()**
- **ProCollectioninstrRefAdd()**
- **ProCollectionInstrAdd()**
- **ProCollectionInstrVisit()**
- **ProCollectioninstrTypeGet()**
- **ProCollectioninstrRefVisit()**
- **ProCollectionrefSelectionGet()**
- **ProCollectionrefTypeGet()**
- **ProCollectionFree()**
- **ProCollectioninstrFree()**
- **ProCollectionrefFree()**
- **ProCollectionInstrRemove()**
- **ProCollectioninstrRefRemove()**
- **ProCrvcollectionInstrRegen()**

The functions listed in this section have been superseded. PTC recommends using the `ProCrvcollection*` and `ProSrfcollection*` functions instead of these; documentation for these functions is provided for maintenance of existing applications.

Example 1: Interactive Curve Collection using Creo Parametric TOOLKIT

The sample code in the file `UgGeomCurveLength.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_geometry` demonstrates the interactive curves collection and computes the total curve length - consisting of addition of lengths of individual curves in resulting selection set. The collection filter function is set to allow only edge type of selections.

Example 2: Interactive Surface Collection using Creo Parametric TOOLKIT

The sample code in the file `UgGeomSurfArea.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_geometry` demonstrates the interactive surface collection and computes the total

area - consisting of addition of areas of individual surfaces in resulting selection set. It also creates parameter of double type on the individual surfaces and assigns it with the value of the individual surface area.

23

User Interface: Animation

This chapter describes the Creo Parametric TOOLKIT functions that enable you to create animation frames and movies.

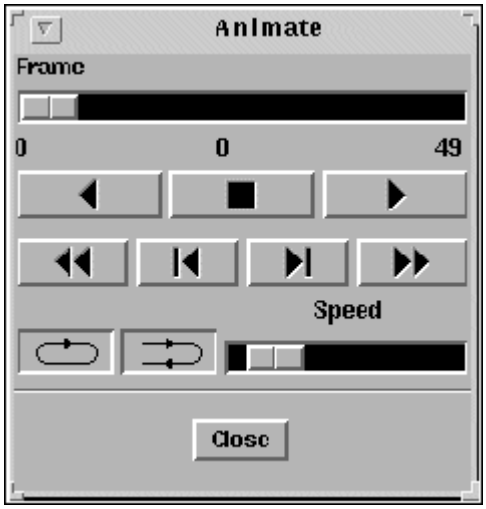
Introduction

Creo Parametric TOOLKIT provides the functions that enable you to animate parts and assemblies. Creo Parametric TOOLKIT handles lower-level considerations, such as hardware-dependent graphics and the user interface.

Two animation techniques are available:

- Batch animation—You create an animation movie (`ProAnimMovie` object) and users control the movie using an interface similar to a VCR. Users can perform such operations as “Play”, “Fast-Forward”, “Rewind”, and so on. The following figure shows the animation dialog box.

  Animation Dialog Box

- Frame-by-frame (single) animation—You create a single animation (`ProSingleAnim` object) and code the control loop into your application. The batch animation interface does not appear. To replay the movie, you must reexecute the control loop of your application.

For both batch and single animation, you must build the animation from two important elements. These elements are:

- Animation object (`ProAnimObj` object)—Contains the object to be animated and its transformation, with respect to its immediate parent. In constructing the animation, you can show motion by creating a series of `ProAnimObj` objects, each with a different transformation.
- Animation frame (`ProAnimFrame` object)—Captures the image of the animation objects at one instance of the animation. Your final animation is a series of animation frames shown in succession.

Animation Objects

Functions Introduced:

- **ProAnimobjectCreate()**
- **ProAnimobjectDelete()**

An animation object can be a part or an assembly. To create an animation object, call the function `ProAnimobjectCreate()` and pass as input the component path of the object to be animated. You must also supply the location of the animation object with respect to its immediate parent—not with respect to the top-level assembly. This distinction is important when the depth of the assembly hierarchy is greater than 1.

To delete an animation object, use the function `ProAnimobjectDelete()`.

Animation Frames

Functions Introduced:

- **ProAnimframeCreate()**
- **ProAnimframeObjAdd()**
- **ProAnimframeObjRemove()**
- **ProAnimframeDelete()**
- **ProAnimframeObjVisit()**

After you have created animation objects, you must create an animation frame in which to store the objects. To create an animation frame, call the function `ProAnimframeCreate()` and supply a frame view matrix. The frame view matrix is a transformation from the top model coordinate system that allows you to alter the view of the top-level model in your animation. This functionality could be used, for example, to change the view of an assembly while the assembly components (animation objects) move as specified in the call to `ProAnimobjectCreate()`.

Note

The frame view matrix is specified as a transformation from the root assembly.

Creating an animation frame does not cause the animation objects to be contained in the frame. To add animation objects to a frame, you must call the function `ProAnimframeObjAdd()`.

To remove an object from a frame, call the function `ProAnimframeObjRemove()`. To delete a frame, call `ProAnimframeDelete()`.

The function `ProAnimframeObjVisit()` enables you to visit each animation object in an animation frame. The input arguments of the function specify the action and filtration functions, which are of type `ProAnimObjAct`.

Playing Animations

This section describes how to use your animation frames to construct and play the animation. As previously mentioned, there are two types of animation—single and batch.

Single Animation

Functions Introduced:

- **ProSingleAnimationInit()**
- **ProSingleAnimationPlay()**
- **ProSingleAnimationClear()**

If you want to use single animation, your Creo Parametric TOOLKIT application must include a control loop that displays one animation frame after another. Before executing your control loop, initialize the single animation by calling `ProSingleAnimationInit()`. Within the loop, display each frame in turn using the function `ProSingleAnimationPlay()`.

Note

Single animation does not involve the batch animation user interface. The control over a single animation is contained entirely within your application.

The function `ProSingleAnimationClear()` clears the specified single animation.

Batch Animation

Batch animation implements the user interface shown in the Introduction on page 536 section. The interface enables users to control the playing of your animation movie.

Animation Movies

Functions Introduced:

- **ProAnimmovieCreate()**
- **ProAnimmovieFrameAdd()**
- **ProAnimmovieFrameRemove()**
- **ProAnimmovieDelete()**
- **ProAnimmovieFrameVisit()**

The function `ProAnimmovieCreate()` creates an animation movie. At its creation, a movie does not contain any frames. To add frames to the animation movie, call the function `ProAnimmovieFrameAdd()`.

To remove a frame from an animation movie, call the function `ProAnimmovieFrameRemove()`. Note that this action does not cause the frame to be deleted; use `ProAnimmovieDelete()` to release the memory of the animation frame.

The function `ProAnimmovieFrameVisit()` enables you to visit each of the frames in an animation movie. The input arguments to the function specify the action and filtration functions, which are of type `ProAnimFrameAct`.

Playing a Batch Animation

Function Introduced:

- **ProBatchAnimationStart()**
- **ProAnimmovieSpinflagSet()**

Batch animation manages the display of animation frames inside Creo Parametric. When you call `ProBatchAnimationStart()`, the system displays the VCR-like user interface. This interface enables users to control the speed and direction of the animation.

The function `ProBatchAnimationStart()` requires as input the animation movie to be started (animated). In addition, you can supply a callback function to be invoked before each animation frame is displayed. The callback function is of type `ProBatchAnimAct`.

Use the function `ProAnimmovieSpinflagSet()` to set the `ProBoolean` flag that allows spin in a batch animation process. If the animation includes view modifications, this flag should be set to false, otherwise it can be true.

Example 1: Creating a Batch Animation

The sample code in the file `UgAnimAsmcompAnim.c` located at `<creo_toolkit_loadpoint>\protk_appls\pt_userguide\ptu_anim`, shows how to animate an assembly component. The selected component rotates about the x-axis.

24

Annotations: Annotation Features and Annotations

This chapter describes how to access annotation features for special customizations. It provides specific functions for creation, access, and modification of annotation features and elements.

Overview of Annotation Features

An annotation feature is a new feature available in Pro/ENGINEER Wildfire 2.0. It is composed of one or more "annotation elements". Each annotation element is composed of references, parameters and annotations (notes, symbols, geometric tolerances, surface finishes, reference dimensions, driven dimensions, and manufacturing template annotations). The annotation feature allows annotation information to have the same benefits as geometry in Creo Parametric models, that is, parameters can be assigned to these annotation elements, and missing geometric references can cause features to fail in some situations.

The feature type `PRO_FEAT_ANNOTATION` represents an annotation feature. Functions referring to annotation features use the structure `ProAnnotationfeat`, which is identical to `ProFeature`.

Functions referring to annotation elements use the structure `ProAnnotationElem` which is identical to the structure `ProModelitem` and is defined as

```
typedef struct pro_model_item
{
   ProType type;
   int id;
   ProMdl owner;
}ProAnnotationElem
```

Like other `ProModelitem` derivatives, each annotation element has a unique id assigned to it in the model.

Annotation elements may belong to annotation features, or may also be found in data-sharing features (features like Copy Geometry, Publish Geometry, Merge, Cutout, and Shrinkwrap features).

Creo Parametric TOOLKIT does not expose the feature element tree for annotation features because some elements in the tree are used for non-standard purposes. Instead, Creo Parametric TOOLKIT provides specific functions for creating, redefining, and reading the properties of annotation features and annotation elements.

Creating Annotation Features

Functions Introduced:

- **ProAnnotationfeatCreate()**
- **ProDatumtargetAnnotationfeatureCreate()**

The function `ProAnnotationfeatCreate()` creates a new annotation feature in the model. Specify the following as the input parameters for this function:

- *model*—Specify the solid model on which the feature will be created. Specify the component path if the feature is created in an assembly context.
- *use_ui*—Specifies a boolean flag that determines how the annotation features will be created. It can have the following values:
  - `FALSE`—Indicates that the feature will be a new empty annotation feature with one general annotation element in it. Modify the new annotation element and add others using subsequent steps.
  - `TRUE`—Indicates that Creo Parametric will invoke the annotation feature creation user interface.

The function `ProDatumtargetAnnotationfeatureCreate()` creates a new Datum Target Annotation Feature (DTAF) in the model. This function takes the same input arguments as the earlier function `ProAnnotationfeatCreate()`.

Redefining Annotation Features

Redefining an annotation feature involves creation of new annotation elements, deletion of elements that are not required and modification of the feature properties.

Note

The functions in this section are shortcuts to redefining the feature containing the annotation elements. Because of this, Creo Parametric must regenerate the model after making the required changes to the annotation element. The functions include a flag to optionally allow the **Fix Model** User Interface to appear upon a regeneration failure.

Functions Introduced:

- **ProAnnotationfeatElementAdd()**
- **ProAnnotationfeatElementArrayAdd()**
- **ProAnnotationfeatElementDelete()**
- **ProAnnotationfeatElementCopy()**

The function `ProAnnotationfeatElementAdd()` adds a new non-graphical annotation element to the feature.

The function `ProAnnotationfeatElementArrayAdd()` adds a series of new annotation elements to the feature. Each element may be created as non-graphical or may be assigned a pre-existing annotation.

Note

In case of Datum Target Annotation Features (DTAFs), you can add only one set datum tag annotation element using the function `ProAnnotationfeatElementArrayAdd()`.

The function `ProAnnotationfeatElementDelete()` deletes an annotation element from the feature. The function deletes the annotation element, its annotations, parameters, references, and application data from the feature.

Note

In case of Datum Target Annotation Features (DTAFs), `ProAnnotationfeatElementDelete()` allows you to delete only a Datum Target Annotation Element (DTAE) from a DTAF. This function does not allow deletion of a set datum tag annotation element from the DTAF.

The function `ProAnnotationfeatElementCopy()` copies and adds an existing annotation element to the specified annotation feature.

Visiting Annotation Features

Functions Introduced:

- **ProSolidFeatVisit()**
- **ProModelitemNameGet()**

Use the function `ProSolidFeatVisit()` to visit the annotation features in the specified model.

The function `ProModelitemNameGet()` returns the name of the annotation feature.

Creating Datum Targets

Functions Introduced:

- **ProMdlDatumTargetCreate()**

The function `ProAnnotationfeatDatumtargetCreate()` has been deprecated. Use the function `ProMdlDatumTargetCreate()` instead. The function `ProMdlDatumTargetCreate()` creates a new datum target. The input arguments are:

- *p_owner_mdl*—Specifies the model in which the datum target will be created.
- *type*—Specifies the type of target area using the enumerated data type `ProDatumTargetType`. The valid values are:
  - `PRO_DATUM_TARGET_NONE`
  - `PRO_DATUM_TARGET_POINT`
  - `PRO_DATUM_TARGET_CIRCLE`
  - `PRO_DATUM_TARGET_RECTANGLE`
- *annot_plane*—Specifies the annotation plane.
- *attach_sels*—Specifies the reference to which the datum target will be attached. To specify a single reference, pass `ProSelection` for index `0` and `NULL` for index `1`.

  For a pair of references, pass `ProSelection` for both indexes. In this case, the datum target is attached to the solid at the intersection point of the two references.
- *text_pnt*—Specifies the location of the text in the datum target.

Visiting Annotation Elements

The functions described in this section enable you to visit all the annotation elements in a solid model.

Functions Introduced:

- **ProFeatureAnnotationelemsVisit()**
- **ProSolidAnnotationelemsVisit()**

The function `ProFeatureAnnotationelemsVisit()` visits the annotation elements in the specified feature.

The function `ProSolidAnnotationelemsVisit()` visits the annotation elements in a solid model.

The filter function `ProAnnotationelemFilterAction()` and the visit function `ProAnnotationelemVisitAction()` are specified as input arguments for the above functions.

The function `ProAnnotationelemFilterAction()` is a generic function for filtering an annotation element. It returns the filter status of the specified annotation element. The filter status is passed as the input argument to the function `ProAnnotationelemVisitAction()` which is a generic function for visiting annotation elements.

Accessing Annotation Elements

The following functions provide access to the properties of an annotation element.

Functions Introduced:

- **ProAnnotationelemAnnotationGet()**
- **ProAnnotationelemCopyGet()**
- **ProAnnotationelemFeatureGet()**
- **ProAnnotationelemIsDependent()**
- **ProAnnotationelemIsIncomplete()**
- **ProAnnotationelemReferencesCollect()**
- **ProAnnotationelemQuiltreferenceSurfacesCollect()**
- **ProAnnotationelemTypeGet()**
- **ProAnnotationelemReferenceDescriptionGet()**
- **ProAnnotationelemReferenceIsStrong()**
- **ProAnnotationelemReferenceAutopropagateGet()**
- **ProAnnotationelemHasMissingrefs()**

The function `ProAnnotationelemAnnotationGet()` returns the annotation contained in an annotation element.

The function `ProAnnotationelemCopyGet()` returns the copy flag of the annotation elements. This property is not supported for elements in data sharing features.

The function `ProAnnotationelemFeatureGet()` returns the feature that owns the annotation element.

The function `ProAnnotationelemIsDependent()` returns the value of the dependency flag for the annotation element. This property is supported only for the elements in data sharing features.

The function `ProAnnotationelemIsIncomplete()` returns a true value if the annotation element has missing strong references.

The function `ProAnnotationelemReferencesCollect()` returns an array of references contained in the specified annotation element. The input arguments for this function are:

- *element*—Specifies the annotation element.
- *ref_type*—Specifies the type of references and can have one of the following values:
  - PRO_ANNOTATION_REF_ALL—All references
  - PRO_ANNOTATION_REF_WEAK—Weak references
  - PRO_ANNOTATION_REF_STRONG—Strong references
- *source*—Specifies the source of the references and can have one of the following values:
  - PRO_ANNOTATION_REF_ALL—From both user and annotation.
  - PRO_ANNOTATION_REF_FROM_ANNOTATION—From the annotation (or custom data) only.
  - PRE_ANNOTATION_REF_FROM_USER—From the user only.

Annotation elements have special default behavior for propagation of datum points to features in other models. The flag that controls this behavior can automatically propagate datum points or any other applicable items to data sharing features after the user has selected all other strong references of the annotation elements.

Here, applicable items are items that are designated to auto-propagate, using a checkbox in the references collector, for specific annotation elements.

The function ProAnnotationelemQuiltreferenceSurfacesCollect() returns the surfaces which make up a quilt surface collection reference for the annotation element.

Note

All the surfaces made inactive by features occurring after the annotation element in the model regeneration are also returned.

The function ProAnnotationelemReferenceAutopropagateGet() gets the autopropagate flag of the specified annotation element reference.

The function ProAnnotationelemTypeGet() returns the type of the annotation contained in the annotation element. It can have one of the following values:

- PRO_ANNOT_TYPE_NONE—Specifies a non-graphical annotation.
- PRO_ANNOT_TYPE_NOTE—Specifies a note. Refer to the section Notes on page 597 for details.

- `PRO_ANNOT_TYPE_GTOL`—Specifies a geometric tolerance. Refer to the chapter Geometric Tolerances on page 606 for details.
- `PRO_ANNOT_TYPE_SRFFIN`—Specifies a surface finish. Refer to the section Surface Finish Annotations on page 612 for details.
- `PRO_ANNOT_TYPE_SYMBOL`—Specifies a symbol. Refer to the section Symbol Annotations on page 614 for details.
- `PRO_ANNOT_TYPE_DRVDIM` —Specifies a driven dimension. Refer to the section Accessing Reference and Driven Dimensions on page 590 for details.
- `PRO_ANNOT_TYPE_REFDIM`—Specifies a reference dimension. Refer to the section Accessing Reference and Driven Dimensions on page 590 for details.
- `PRO_ANNOT_TYPE_CUSTOM`—Specifies a manufacturing template annotation.
- `PRO_ANNOT_TYPE_SET_DATUM_TAG`—Specifies a set datum tag. Refer to the section Accessing Set Datum Tags on page 606 for details.
- `PRO_ANNOT_TYPE_DRIVINGDIM`—Specifies a driving dimension annotation element. Refer to the section Driving Dimension Annotation Elements on page 590 for details.

The function `ProAnnotationelemReferenceDescriptionGet()` gets the description property for a given annotation element reference.

Note

The description string is same as that of the tooltip text for the reference name in the Annotation Feature UI.

The function `ProAnnotationelemReferenceIsStrong()` identifies if a reference is weak or strong in a given annotation element.

The function `ProAnnotationelemHasMissingrefs()` enables you to identify if an annotation element has missing references. The input parameters of this function allow you to investigate specific types and sources of references, or check all references simultaneously.

Modification of Annotation Elements

The functions described in this section allow you to modify the properties of an annotation element.

Note

The functions in this section are shortcuts to redefining the feature containing the annotation elements. Because of this, Creo Parametric must regenerate the model after making the indicated changes to the annotation element. The functions include a flag to optionally allow the **Fix Model** User Interface to appear upon a regeneration failure.

Functions Introduced:

- **ProAnnotationelemAnnotationSet()**
- **ProAnnotationelemCopySet()**
- **ProAnnotationelemDependencySet()**
- **ProAnnotationelemReferenceAdd()**
- **ProAnnotationelemReferenceRemove()**
- **ProAnnotationelemReferenceStrengthen()**
- **ProAnnotationelemReferenceWeaken()**
- **ProAnnotationelemReferenceDescriptionSet()**
- **ProAnnotationelemReferenceAutopropagateSet()**
- **ProAnnotationelemReferencesSet()**
- **ProAnnotationelemArrayReferencesSet()**
- **ProAnnotationreferencesetAlloc()**
- **ProAnnotationreferencesetReferenceAdd()**
- **ProAnnotationreferencesetFree()**
- **ProAnnotationreferencesetProarrayFree()**

The function `ProAnnotationelemAnnotationSet()` allows you to modify the annotation contained in an annotation element. Specify the value for the input argument *annotation* as `NULL` to modify the annotation element to be a non-graphical annotation.

Note

The above function does not support Datum Target Annotation Elements (DTAEs).

If you modify the annotation contained in the annotation element, the original annotation is automatically removed from the element and is owned by the model.

The function `ProAnnotationelemCopySet()` provides write access to the copy flag of the annotation element. This property is not supported for annotations in data sharing features.

The function `ProAnnotationelemDependencySet()` sets the value for the dependency flag. This property is supported only for annotations in data sharing features.

The function `ProAnnotationelemReferenceAdd()` adds a strong user-defined reference to the annotation element.

The function `ProAnnotationelemReferenceRemove()` removes the user defined reference from the annotation element.

The function `ProAnnotationelemReferenceStrengthen()` converts a weak reference to a strong reference.

The function `ProAnnotationelemReferenceWeaken()` converts a strong reference to a weak reference.

The function `ProAnnotationelemReferenceDescriptionSet()` sets the description property for a given annotation element reference.

The function `ProAnnotationelemReferenceAutopropagateSet()` sets the autopropagate flag of the specified annotation element reference.

The function `ProAnnotationelemReferencesSet()` replaces all the user-defined references in the annotation element with a `ProArray` of references specified as the input argument.

The function `ProAnnotationelemArrayReferencesSet()` replaces all the user-defined references for a `ProArray` of annotation elements with the `ProArray` of reference sets specified as the input argument.

Note

All the annotation elements must belong to the same feature. The number of reference sets should match the number of annotation elements to be modified.

The function `ProAnnotationreferencesetAlloc()` allocates a set of user-defined references to be assigned to an annotation element.

The function `ProAnnotationreferencesetReferenceAdd()` adds a new reference to an existing set of user-defined annotation references.

The function `ProAnnotationreferencesetFree()` releases the set of user-defined references to be assigned to an annotation element.

The function `ProAnnotationreferencesetProarrayFree()` releases an array of reference sets to be assigned to an annotation element.

Parameters Assigned to Annotation Elements

The functions described in this section allow you to access parameters assigned to the annotation element. Specify the annotation element as the parameter owner for these functions.

Functions Introduced:

- **ProParameterIsEnumerated()**
- **ProParameterRangeGet()**

The function `ProParameterIsEnumerated()` identifies an enumerated parameter and returns the available values assigned to it. The output of the function is `PRO_B_TRUE` if the parameter is enumerated and `PRO_B_FALSE` if it is not. The output argument also presents `ProArray` of values that can be assigned to the enumerated parameter.

The function `ProParameterRangeGet()` identifies whether a parameter is restricted to a range of values. It optionally provides the boundary conditions for the range.

For more information on functions that allow you to view, create, delete, and modify parameters, refer to Core: Parameters on page 210.

Automatic Propagation of Annotation Elements

Automatic local propagation of annotation elements can be done based on some specified conditions, using a Creo Parametric TOOLKIT application. A Creo Parametric TOOLKIT application can register the following notification event types (`ProNotifyType`):

- `PRO_FEATURE_CREATE_POST`
- `PRO_FEATURE_REDEFINE_POST`

When an appropriate event occurs during a Creo Parametric session, the associated notification function can invoke a local propagation.

Functions Introduced:

- **ProAnnotationelemAutopropagate()**

The function `ProAnnotationelemAutopropagate()` causes the local and automatic propagation of annotation elements to the currently selected feature within the same model, after a supported feature has either been created or modified. The propagation behavior is dependant on the standard Creo Parametric algorithm and on the current contents of the selection buffer.

Following are the list of supported features:

- Draft

    - Surface
    - Solid
- Offset
    - Surface
    - With Draft
    - Expand
- Mirror Surface
- Copy Surface
- Move Surface

The Creo Parametric TOOLKIT application can be written so as to specify the condition for the automatic propagation, based on created feature-type, subtype or any other required properties.

The function propagates based on the current contents of the selection buffer.

For notification of type `PRO_FEATURE_REDEFINE_POST`, Creo Parametric does not automatically populate the selection buffer with the feature that was redefined. The Creo Parametric TOOLKIT application would be then required to populate the selection buffer with the feature using the appropriate functions. Refer to the section Selection Buffer on page 512 in the chapter User Interface: Selection on page 503 for more information on selection buffer.

Note

The function `ProAnnotationelemAutopropagate()` works regardless of the current value for the configuration option, `auto_propagate_ae`. PTC advises that the Creo Parametric TOOLKIT application respect the current value of this configuration option; otherwise, duplicate versions of the propagated annotations can result.

Detail Tree

Creo Parametric 1.0 onwards, when the 3D **Annotation** tab is active, you can view the active combination state of a model and the annotations assigned to it. This representation is called a Detail Tree. For more information on Detail Tree, see the Creo Parametric help. Use the following functions to refresh, expand, and collapse the detail tree:

Functions Introduced:

- **ProDetailtreeRefresh()**
- **ProDetailtreeExpand()**
- **ProDetailtreeCollapse()**

Use the function `ProDetailtreeRefresh()` to rebuild the detail tree in the Creo Parametric window that contains the model.

Use the function `ProDetailtreeExpand()` to expand the detail tree in the Creo Parametric window.

Use the function `ProDetailtreeCollapse()` to collapse all nodes of the detail tree in the Creo Parametric window and make its child nodes invisible.

The input arguments to these functions are:

- *solid*—Handle to the model that contains the detail tree.
- *window_id*—ID of the Creo Parametric window in which you want to refresh, expand, or collapse the detail tree.

Note

Use `PRO_VALUE_UNUSED` to refresh, expand, or collapse the detail tree in the active window.

Access to Annotations

The structure for the annotations is similar to the structure `ProModelitem` and is defined as

```
typedef struct pro_model_item
{
   ProType type;
   int id;
   ProMdl owner;
}ProAnnotation
```

The value of *type* for different annotations is as follows:

- `PRO_NOTE`—Specifies a note. Functions specific to notes use the object type `ProNote`.
- `PRO_SYMBOL_INSTANCE`—Specifies a symbol instance. Functions specific to symbols use the object `ProDtlsyminst`.
- `PRO_GTOL`—Specifies a geometric tolerance. Functions specific to Gtols use the object `ProGtol`.
- `PRO_SURF_FIN`—Specifies a surface finish. Functions specific to surface finish use the object `ProSurfFinish`.

- PRO_REF_DIMENSION—Specifies a reference dimension.
- PRO_DIMENSION—Specifies a driving or driven dimension. Reference, driven and driving dimension functions may use the object type ProDimension.
- PRO_SET_DATUM_TAG—Specifies a set datum tag annotation. Functions specific to set datum tag use the object type ProSetdatumtag.
- PRO_CUSTOM_ANNOTATION—Specifies a custom annotation type. Currently, used only for manufacturing template annotations.

Functions Introduced:

- **ProAnnotationElementGet()**
- **ProAnnotationByViewShow()**
- **ProAnnotationByFeatureShow()**
- **ProAnnotationByComponentShow()**
- **ProCombstateAnnotationErase()**
- **ProDrawingAnnotationErase()**
- **ProAnnotationShow()**
- **ProAnnotationIsShown()**
- **ProAnnotationIsInactive()**
- **ProAnnotationDisplay()**
- **ProAnnotationUndisplay()**
- **ProAnnotationUpdate()**
- **ProFeatureParamsDisplay()**

The function ProAnnotationElementGet() returns the annotation element that contains the annotation.

The function ProAnnotationByViewShow() displays the annotation of the specified type in the selected view.

The function ProAnnotationByFeatureShow() displays the annotation of the specified type for the selected feature.

The function ProAnnotationByComponentShow() displays the annotation of the specified type for the selected component.

From Creo Parametric 2.0 M060 onward, you can specify the view in which the annotations for the selected feature and component must be displayed. For the functions ProAnnotationByFeatureShow() and

ProAnnotationByComponentShow(), specify the view where the annotations must be displayed. If you pass the input argument *view* as NULL, the annotations are displayed in all the views.

Note

In Creo Parametric 2.0 M050, you must pass the input argument *view* as NULL.

The following types of annotations are displayed for the functions ProAnnotationByViewShow(), ProAnnotationByFeatureShow(), and ProAnnotationByComponentShow():

- PRO_DIMENSION
- PRO_REF_DIMENSION
- PRO_NOTE
- PRO_GTOL
- PRO_SURF_FIN
- PRO_AXIS
- PRO_SET_DATUM_TAG
- PRO_SYMBOL_INSTANCE
- PRO_DATUM_TARGET

If you want to display an annotation which is dependent on another annotation for its display in the drawing, then the Creo Parametric TOOLKIT application must first display the parent annotation. Only after the parent annotation is displayed, the application can display its dependent annotations. For example, if a geometric tolerance is placed on a dimension, then the application must call the function ProAnnotationByViewShow() for PRO_DIMENSION type. The dimension is displayed. To display the geometric tolerance, call the function ProAnnotationByViewShow() for PRO_GTOL. The same logic applies for the functions ProAnnotationByFeatureShow(), and ProAnnotationByComponentShow().

The function ProCombstateAnnotationErase() removes an annotation from the display for the specified combined state.

The function ProDrawingAnnotationErase() removes an annotation from the display for the specified drawing. The annotation is not shown in the specified drawing.

Note

The annotation which was removed from the display using the functions `ProCombstateAnnotationErase()` and `ProDrawingAnnotationErase()` will become visible again, only if the function `ProAnnotationShow()` is called to explicitly display the annotation.

The function `ProAnnotationShow()` shows the annotation in the current combined state. The annotation will be visible until it is explicitly erased from the combined state . The function also adds the specified annotation to the current combined state, if not added. If the specified annotation has been erased, then the function changes the display status of the erased annotation and makes it visible in the combined state, that is, it unerases the annotation.

This function is also capable of showing an annotation in a drawing view similar to the Creo Parametric command `Show` and `Erase`. This function supersedes the functions `ProDimensionShow()` and `ProNoteDisplay()`.

The function `ProAnnotationIsShown()` returns the display status of the annotation in the current combined state or drawing.

Note

To get the display status of set datum tags in a drawing, use the function `ProDrawingSetDatumTagIsShown()`.

The function `ProAnnotationIsInactive()` indicates whether the annotation can be shown or not. An annotation becomes inactive if all the weak references it points to have been lost or consumed.

The functions `ProAnnotatonDisplay()` and `ProAnnotationUndisplay()` temporarily display and remove an annotation from the display for the specified combined state or drawing. The functions `ProAnnotatonDisplay()` and `ProAnnotationUndisplay()` should be used together. To edit a shown annotation, it must be first removed temporarily from display using the function `ProAnnotationUndisplay()` followed by the editing function calls, and finally must be redisplayed using the function `ProAnnotationDisplay()`, so that the updated annotation is correctly visible to the user.

The function `ProAnnotationUpdate()` updates the display of an annotation, but does not actually display it. If the annotation is not currently displayed (because it is hidden by layer status or inactive geometry), the text extracted from the annotation with the mode `PRODISPMODE_NUMERIC` may include callout

symbols, instead of the text shown to the user. `ProAnnotationUpdate()` informs Creo Parametric to update the contents of the annotation in order to cross-reference these callout values. This function supports 3D model notes of the type `ProNote` and detail notes of the type `ProDtlnote`.

When you want to force the display of dimensions or parameters, geometric tolerances (gtols), and so on on a single feature, use the function `ProFeatureParamsDisplay()`.

Note

- The function `ProAnnotationDisplay()` supersedes the functions `ProDimensionShow()` and `ProGtolShow()`.
- The function `ProAnnotationUndisplay()` supersedes the functions `ProDimensionErase()` and `ProGtolErase()`.
- The function `ProDimensionDisplayUpdate()` is superseded by a combination of `ProAnnotationDisplay()` and `ProAnnotationUndisplay()`.

Converting Annotations to Latest Version

Functions Introduced:

- **ProAnnotationNeedsConversion()**
- **ProAnnotationLegacyConvert()**

The function `ProAnnotationNeedsConversion()` returns true in the following cases:

- Annotations created in releases earlier than Creo Parametric 4.0 F000
- Annotations created using the deprecated functions `ProGtolCreate()` or `ProSetdatumtagCreate()`

The input argument *annotation* can be a gtol, reference dimension, driven dimension, set datum tag, or datum tag which needs to be converted.

The function returns the following values for the output argument `needs_conversion`:

- `PRO_B_TRUE`—When the annotation needs conversion.
- `PRO_B_FALSE`—When the annotation is already converted.

The function `ProAnnotationLegacyConvert()` converts annotations to the latest Creo Parametric version.

You can call the function `ProAnnotationLegacyConvert()` only if the function `ProAnnotationNeedsConversion()` returns true.

Annotation Text Styles

Functions Introduced:

- **ProAnnotationTextstyleGet()**
- **ProAnnotationTextstyleSet()**
- **ProTextStyleFree()**

The function `ProAnnotationTextstyleGet()` retrieves the text style for the specified annotation. The input arguments are:

- *annotation*—Specifies the annotation.
- *drawing*—Specifies a drawing only when the annotation is owned by the solid, but is displayed in the drawing.
- *comp_path*—Specifies the component path to the solid that owns the annotation.

Use the function `ProTextStyleFree()` to free the allocated data structure.

The method `ProAnnotationTextstyleSet()` sets the text style for the specified annotation.

Annotation Orientation

An Annotation Orientation refers to the annotation plane or the parallel plane in which the annotation lies, the viewing direction, and the text rotation. You can define the annotation orientation using a datum plane or flat surface, a named view, or as flat to screen. If the orientation is defined by a datum plane, you can set its reference to `frozen` or `driven`; where `frozen` indicates that the reference to the datum plane or named view has been removed.

Functions Introduced:

- **ProAnnotationplaneCreate()**
- **ProAnnotationplaneFromViewCreate()**
- **ProAnnotationplaneFlatToScreenCreate()**
- **ProAnnotationplaneTypeGet()**
- **ProAnnotationplaneReferenceGet()**
- **ProAnnotationplanePlaneGet()**
- **ProAnnotationplaneVectorGet()**
- **ProAnnotationplaneFrozenGet()**

- **ProAnnotationplaneFrozenSet()**
- **ProAnnotationplaneForcetoplaneflagGet()**
- **ProAnnotationplaneForcetoplaneflagSet()**
- **ProAnnotationplaneViewnameGet()**
- **ProAnnotationplaneAngleGet()**
- **ProAnnotationplaneActiveGet()**
- **ProMdlAnnotplanesFromGalleryCollect()**
- **ProMdlAnnotationplanesCollect()**
- **ProAnnotationplaneNamesGet()**
- **ProAnnotationplaneByNameInit()**
- **ProAnnotationplaneNameAssign()**
- **ProAnnotationplaneAddToGallery()**
- **ProAnnotationplaneRemoveFromGallery()**
- **ProAnnotationRotate()**

The function `ProAnnotationplaneCreate()` creates a new annotation plane from either a datum plane, a flat surface, or an existing annotation that already contains an annotation plane.

The function `ProAnnotationplaneFromViewCreate()` creates a new annotation plane from a saved model view.

The function `ProAnnotationplaneFlatToScreenCreate()` returns the annotation plane item representing a flat-to-screen annotation in the model. This function takes a `ProBoolean` input argument *by_screen_point*, which identifies whether the annotations on this plane are located by screen points, or by model units.

Note

You can only place notes, surface finishes, and symbols as flat to screen. Dimensions, geometric tolerances and set datum tags are not supported as flat-to-screen annotations.

Use the function `ProAnnotationplaneTypeGet()` to obtain the annotation plane type. It can have one of the following values:

- PRO_ANNOTATIONPLANE_REFERENCE—The annotation plane is created from a datum plane or a flat surface, and can be frozen or be associative to the reference.
- PRO_ANNOTATIONPLANE_NAMED_VIEW—The annotation plane is created from a named view or a view in the drawing.
- PRO_ANNOTATIONPLANE_FLATTOSCREEN_BY_MODELPNT—The annotation plane is flat-to-screen and annotations are located by model units.
- PRO_ANNOTATIONPLANE_FLATTOSCREEN_BY_SCREENPNT—The annotation plane is flat-to-screen and annotations are located by screen points.
- PRO_ANNOTATIONPLANE_FLATTOSCREEN_LEGACY—The annotation uses a legacy flat-to-screen format (located in model space).

The function ProAnnotationplaneReferenceGet() returns the planar surface used as the annotation plane.

The function ProAnnotationplanePlaneGet() returns the geometry of the annotation plane in terms of the ProPlanedata object containing the origin and orientation of the annotation plane.

The functions ProAnnotationplaneFrozenGet() and ProAnnotationplaneFrozenSet() determine and assign, respectively, whether the annotation orientation is frozen or driven by reference to the plane geometry. These functions are applicable only for annotation planes governed by references.

The functions ProAnnotationplaneForcetoplaneflagGet() and ProAnnotationplaneForcetoplaneflagSet() return and assign, respectively, the value of the ProBoolean argument *force_to_plane* for an annotation plane. If this argument is set to PRO_B_TRUE, then the annotations that reference the annotation plane are placed on that plane. If the annotation orientation is not frozen, that is, driven by the reference plane, and if the reference plane is moved, then the annotations also move along with the plane.

The function ProAnnotationplaneViewnameGet() obtains the name of the view that was originally used to determine the orientation of the annotation plane.

Note

If the named view orientation has been changed after the annotation plane was created, the orientation of the plane will not match the current orientation of the view.

The function ProAnnotationplaneVectorGet() returns the normal vector that determines the viewing direction of the annotation plane.

The function ProAnnotationplaneAngleGet() returns the current rotation angle in degrees for a given annotation plane and the text orientation of all annotations on that plane.

The function ProAnnotationplaneActiveGet() returns the active annotation plane in the specified model.

The function ProMdlAnnotplanesFromGalleryCollect() collects the names of all the annotation planes in the gallery. The output argument names is a ProArray of names in the gallery. Use the function ProWstringArrayFree() to free the allocated memory. The function returns the error PRO_TK_EMPTY if there are no annotation planes in the gallery.

The function ProMdlAnnotationplanesCollect() collects the names of all the named annotation planes in the specified model. The function returns the error PRO_TK_EMPTY if there are no annotation planes in the model.

The function ProAnnotationplaneNamesGet() returns the names of the specified annotation plane.

The function ProAnnotationplaneByNameInit() finds and returns the annotation plane with the specified name. The function returns the error PRO_TK_E_NOT_FOUND if the annotation plane with the specified name does not exist.

The function ProAnnotationplaneNameAssign() assigns a name to the specified annotation plane. The function returns the error PRO_TK_E_NOT_FOUND if the specified annotation plane does not exist in the model. The function returns the error PRO_TK_E_FOUND if an annotation plane with the specified name already exists in the model.

Use the function ProAnnotationplaneAddToGallery() to add an annotation plane with the specified name to the gallery.

Use the function ProAnnotationplaneRemoveFromGallery() to remove the annotation plane with the specified name from the gallery.

The function ProAnnotationRotate() rotates a given annotation by the specified angle. This moves the annotation to a new annotation plane with the appropriate rotation assigned. Other annotations on the annotation's current plane are unaffected by this function.

Note

You can only rotate annotations that belong to annotation elements using the above function.

Annotation Associativity

The functions described in this section allow you to ensure associativity between shown annotations in drawings and 3D models. You can independently control the position associativity and attachment associativity of a drawing annotation.

Note

- Drawing annotations can have only uni-directional associativity, that is, changes to the position and attachment of the annotation in the 3D model are reflected for the annotation in the drawing view, but not vice-versa.
- You cannot modify the position associativity and attachment associativity of a drawing annotation from the 3D model.
- You cannot make free, flat-to-screen annotations in a drawing view associative to the annotations in the 3D model.
- Annotation properties such as text, jogs, breaks, skew, witness line clippings, and flip arrow states are not associative.

Functions Introduced:

- **ProAnnotationIsAssociative()**
- **ProAnnotationPositionUpdate()**
- **ProAnnotationAttachmentUpdate()**

The function `ProAnnotationIsAssociative()` identifies if a given annotation in a drawing view is associative to the annotation in the 3D model. It has the following output arguments:

- *assoc_position*—Specifies if the position of the annotation is associative.
- *assoc_attach*—Specifies the attachment associativity. It takes one of the following values:
  - `PRO_ANNOTATTACH_ASSOCIATIVITY_PARTIAL`—Specifies that the drawing annotation is partially associative.
  - `PRO_ANNOTATTACH_ASSOCIATIVITY_FULL`—Specifies that the drawing annotation is fully associative.
- *future_use*—This argument is for future use.

The function `ProAnnotationPositionUpdate()` updates the position of the drawing annotation, and makes it associative to the position of the annotation in the 3D model.

Note

You can update the associative position only for drawing annotations that have their placement based on model coordinates.

The function `ProAnnotationAttachmentUpdate()` updates the attachment of the drawing annotation, and makes it associative to the attachment of the annotation in the 3D model. Associative attachment of an annotation refers to both – its references and its attachment point on its references.

Note

You can update the associative attachment only for drawing annotations that are attached to the geometry.

Annotation Security

The functions described in this section allow you to manage whether an annotation is designated as a security marking. You can independently control the security marking option of a drawing annotation. You cannot copy such annotations. Annotations designated as security markings have the following characteristics:

- Always visible whenever the model is viewed in a product that supports the security markings.
- Listed at the top of the detail tree in an active combined state.
- Shown by an icon in the Detail Tree and Model Tree.

Functions Introduced:

- **ProAnnotationSecuritymarkingSet()**
- **ProAnnotationSecuritymarkingGet()**

Use the function `ProAnnotationSecuritymarkingSet()` to set the security marking option for notes and symbols. The input arguments follow:

- a
- *annotation*—Annotation must be flat to screen, unattached, and standalone note or symbol.
- *is_secure*—Pass a `ProBoolean` value `PRO_B_TRUE` to designate security marking.

Use the function `ProAnnotationSecuritymarkingGet()` to retrieve the security marking option for notes and symbols.

Interactive Selection

Annotation features, annotation elements, and annotations can be selected interactively using `ProSelect()` or can be obtained from the selection buffer using the function `ProSelbufferSelectionsGet()`.

For more information on interactive selection refer to chapter User Interface: Selection on page 503.

Display Modes

Functions Introduced:

- **ProDisplaymodeGet()**
- **ProDisplaymodeSet()**

These functions specify whether the display of dimensions shows symbols or values, and enables you to switch the mode. This is the equivalent to the Creo Parametric command **Switch Dimensions** in the **Relations** dialog box.

Designating Dimensions and Symbols

Function Introduced:

- **ProSymbolDesignate()**
- **ProSymbolUndesignate()**
- **ProSymbolDesignationVerify()**

The function `ProSymbolDesignate()` designates a dimension, dimension tolerance, geometric tolerance or surface finish symbol to the specified model.

The function `ProSymbolUndesignate()` undesignates the dimension, dimension tolerance, geometric tolerance or surface finish symbol from the specified model.

The function `ProSymbolDesignationVerify()` determines if a dimension, dimension tolerance, geometric tolerance or surface finish symbol has been designated to a model.

Dimensions

The ProDimension Object

The `ProDimension` object handle is a DHandle that is equivalent to `ProModelitem`. The owner field can be a solid or a drawing, depending upon where the dimension is stored. Dimensions created in drawing mode may be stored in the drawing or in the solid depending upon the setting of the `config.pro` option `CREATE_DRAWING_DIMS_ONLY`. The type `field` is either `PRO_DIMENSION` or `PRO_REF_DIMENSION`. The `ID` is the integer used to identify the dimension inside Creo Parametric. It corresponds to the numerical part of the default symbol assigned to the dimension when it is created.

The `ProDimension` object also inherits from `ProModelitem`, which means that functions such as `ProModelitemInit()` and `ProSelectionModelitemGet()` can be used for it (`ProDimensionSymbolGet()` and `ProDimensionSymbolSet()` are recommended for this purpose, instead of `ProModelitemNameGet()` and `ProModelitemNameSet()`).

Visiting Dimensions

Functions Introduced:

- **ProSolidDimensionVisit()**
- **ProDrawingDimensionVisit()**
- **ProDimensionSymbolGet()**
- **ProDimensionValueGet()**
- **ProDimensionParentGet()**
- **ProDimensionTypeGet()**
- **ProDimensionNomvalueGet()**
- **ProDimensionIsDisplayRoundedValue()**
- **ProDimensionDisplayRoundedValueSet()**
- **ProDimensionDisplayedValueGet()**
- **ProDimensionOverridevalueGet()**
- **ProDimensionValuedisplayGet()**
- **ProDimensionIsFractional()**
- **ProDimensionDecimalsGet()**
- **ProDimensionDenominatorGet()**
- **ProDimensionIsReldriven()**

- **ProDimensionIsRegenednegative()**
- **ProDimensionBoundGet()**
- **ProDimensionOwnerfeatureGet()**
- **ProDimensionIsAccessibleInModel()**
- **ProDimensionIsSignDriven()**
- **ProDimensionDisplayFormatGet()**
- **ProDimensionOriginSideGet()**
- **ProSelectionDimWitnessLineGet()**

The two visit functions `ProSolidDimensionVisit()` and `ProDrawingDimensionVisit()` conform to the usual style of visit functions. (Refer to section Visit Functions on page 62 in the chapter Fundamentals on page 22.) A dimension is stored in a solid if it is a “shown” dimension, that is, if it was created automatically by Creo Parametric as part of the feature definition. A dimension will also be stored in a solid if it was created in drawing mode while the `config.pro` option `CREATE_DRAWING_DIMS_ONLY` was set to `NO`.

The function `ProDimensionSymbolGet()` returns the symbol (the name) of the specified dimension.

The function `ProDimensionValueGet()` returns the value of the dimension.

Some feature dimensions are dependent on dimensions of other features. To modify the dependent dimension, you must get the parent dimension and modify it. Use the function `ProDimensionParentGet()` to get the parent dimension of the specified dependent dimension. For example, consider a sketch feature, which is used to create an extrude feature. In this case, the section dimensions of the extrude feature depend on the dimensions of the sketch feature. To modify the section dimensions of extrude feature, the dimensions of the sketch feature must be retrieved and modified.

Note

Multiple dimensions may depend on a single parent dimension.

The function `ProDimensionTypeGet()` returns the type of the dimension in terms of the following values:

- `PRODIMTYPE_LINEAR`
- `PRODIMTYPE_RADIUS`
- `PRODIMTYPE_DIAMETER`

- PRODIMTYPE_ANGLE
- PRODIMTYPE_ARC_LENGTH

The function ProDimensionNomvalueGet() returns the nominal value of a dimension. The function returns the nominal value even if the dimension is set to the upper or lower bound. The nominal value is returned in degrees for an angular dimension and in the system of units for other types of dimensions.

Use the function ProDimensionIsDisplayRoundedValue() to determine whether the specified dimension is set to display its rounded off value.

In Creo Parametric TOOLKIT, a rounded off value is a decimal value that contains only the desired number of digits after the decimal point. For example, if a dimension has the stored value 10.34132 and you want to display only two digits after the decimal point, you must round off the stored value to two decimal places. Thus, rounding off converts 10.34132 to 10.34.

Use the function ProDimensionDisplayRoundedValueSet() to set the attribute of the given dimension to display either the rounded off value or the stored value. You can use this function for all dimensions, except angular dimensions created prior to Pro/ENGINEER Wildfire 4.0, ordinate baseline dimensions, and dimensions of type DIM_IPAR_INT. For these dimensions, the functions returns an error status PRO_TK_NOT_VALID.

If you choose to display the rounded off value, the function ProDimensionDisplayedValueGet() retrieves the displayed rounded value of the specified dimension. Otherwise, it retrieves the stored value.

The function ProDimensionOverridevalueGet() returns the override value for a dimension. The default override value is zero.

Note

The override value is available only for driven dimensions.

Use the function ProDimensionValuedisplayGet() to obtain the type of value displayed for a dimension. The valid types are:

- PRO_DIMVALUEDISPLAY_NOMINAL—Displays the actual value of the dimension along with the tolerance value.
- PRO_DIMVALUEDISPLAY_OVERRIDE—Displays the override value for the dimension along with the tolerance value.
- PRO_DIMVALUEDISPLAY_HIDE—Displays only the tolerance value for the dimension.

The function ProDimensionIsFractional() returns whether the dimension is expressed in terms of a fraction rather than a decimal. If the dimension is decimal, the function ProDimensionDecimalsGet() outputs

the number of decimals digits that are significant; if the dimension is fractional, the function `ProDimensionDenominatorGet()` returns the value of the largest possible denominator used to define the fractional value.

The function `ProDimensionIsReldriven()` returns whether the dimension is driven by a relation.

The function `ProDimensionIsRegenednegative()` returns whether the dimension really has a negative value in relation to its original definition. Dimensions are always displayed in Creo Parametric with positive values, and `ProDimensionValueGet()` will always return a positive value, so this function is needed to show whether a dimension has been "flipped" as a result of being assigned a negative value during the last regeneration.

The function `ProDimensionBoundGet()` returns the bound status of a dimension.

The function `ProDimensionFeatureGet()` has been deprecated. Use the function `ProDimensionOwnerfeatureGet()` instead.

The function `ProDimensionOwnerfeatureGet()` returns the feature that owns the specified dimension.

Note

For dimensions or reference dimensions in annotation elements, the function `ProDimensionOwnerfeatureGet()` returns the annotation feature that directly owns the annotation element.

The function `ProDimensionIsAccessibleInModel()` identifies if a specified dimension is owned by the model. By default, the dimension is accessible in the model.

When you set a negative value to a dimension, it will either change the dimension to this negative value, or flip the direction around its reference and show a positive value dimension instead. Use the function `ProDimensionIsSignDriven()` to check this. The function returns the following values for the output argument *is_sign_driven*:

- `PRO_B_TRUE`—When the negative sign in the dimension value is used to flip the direction.
- `PRO_B_FALSE`—When the negative sign is used to indicate a negative value, that is, the dimension is negative.

The configuration option `show_dim_sign` when set to *yes* allows you to display negative dimensions in the Creo Parametric user interface.

When the option is set *no*, the dimensions always show positive value. However, in this case, if you set a negative value for the dimension, the direction is flipped.

Note

Some feature types, such as, dimensions for coordinate systems and datum point offsets, always show negative or positive values, even if the option is set to *no*. These features do not depend on the configuration option.

The function `ProDimensionDisplayFormatGet()` retrieves the format in which the specified dimension is displayed. The enumerated data type `ProDimensionDisplayFormat` returns the following values:

- `PRO_DIM_DISPLAY_DECIMAL`—Specifies that the dimension is displayed in decimal format.
- `PRO_DIM_DISPLAY_FRACTIONAL`—Specifies that the dimension is displayed in fractional format.

For dimensions, sometimes it may be required to indicate the origin or start of measurement. The origin is indicated by placing the dimension origin symbol on the witness line. The function `ProDimensionOriginSideGet()` retrieves the witness line which is set as the origin for a dimension. The output argument *dim_side* returns the index of witness line. If dimension origin has not been set for the specified dimension, the argument returns `-1`.

You can place annotations such as, geometric tolerances and datum feature symbol, on the witness lines of dimensions. The function `ProSelectionDimWitnessLineGet()` gets information about the dimension which has an annotation attached to its witness line. You must get the input object `ProSelection` from the annotation which is attached to the witness line. For example, if the leader of a geometric tolerance is attached to the witness line of a dimension, the `ProSelection` object is returned by the function `ProGtolAttachLeadersGet()`. The output arguments are:

- *dimension*—Specifies a pointer to the dimension which is associated with the selected witness line.
- *wline_side*—Specifies the index of the witness line to which the annotation is attached.
- *location*—Specifies the location on the witness line where the annotation is attached.

Example 2: Changing the Displayed Value of Selected Model Dimension to Rounded or Non-Rounded

The sample code in the file `UgDimDisplayRounded.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dims`, shows the displayed value of a selected model dimension to rounded/non-rounded.

Modifying Dimensions

Functions Introduced:

- **ProDimensionSymbolSet()**
- **ProDimensionValueSet()**
- **ProDimensionOverridevalueSet()**
- **ProDimensionValuedisplaySet()**
- **ProDimensionDecimalsSet()**
- **ProDimensionDenominatorSet()**
- **ProDimensionBoundSet()**
- **ProDimensionDimensionReset()**
- **ProDimensionBasicSet()**
- **ProDimensionInspectionSet()**
- **ProDimensionElbowSet()**
- **ProDimensionArrowtypeSet()**
- **ProDimensionSimpleBreakCreate()**
- **ProDimensionJogCreate()**
- **ProDimensionWitnesslineErase()**
- **ProDimensionWitnesslineShow()**
- **ProDimensionDisplayFormatSet()**
- **ProDimensionOriginSideSet()**
- **ProDimensionEnvelopeGet()**

The function `ProDimensionSymbolSet()` allows you to change the symbol (the name) of a dimension. You can use this function only with solid dimensions.

Note

The function will return the error `PRO_TK_NO_CHANGE` if the specified symbol already exists for another dimension in the model.

The function `ProDimensionValueSet()` changes the value of a dimension. It does not allow you to change the value of any dimension whose value is driven in some other way, for example, a driven or reference dimension or a dimension that is driven by a relation.

The function `ProDimensionOverridevalueSet()` assigns the override value for a dimension. This value is restricted to real numbers. The default override value is zero.

Note

You can set the override value only for driven dimensions.

The function `ProDimensionValuedisplaySet()` sets the type of value to be displayed for a dimension.

The function `ProDimensionDecimalsSet()` sets the number of decimal places for a decimal dimension.

When you call the function `ProDimensionDecimalsSet()` for a driving dimension:

- If the number of decimal places required to display the stored value of the dimension is greater than the number of decimal places specified in the function `ProDimensionDecimalsSet()` and the Round Displayed Value attribute of the dimension is ON, the stored value is unchanged. Only the displayed number of decimal places is changed and the displayed value is updated accordingly.

  For example, consider a dimension with its stored value as 12.12323 and the Round Displayed Value attribute of the dimension is set to ON. If the function `ProDimensionDecimalsSet()` sets the number of decimal places to 3, the stored value of the dimension is unchanged, that is, the stored value will be 12.12323. The displayed value of the dimension is rounded to 3 decimal places, that is, 12.123. The Round Displayed Value attribute is not changed.
- If the number of decimal places required to display the stored value of the dimension is greater than the number of decimal places specified in the function `ProDimensionDecimalsSet()` and the Round Displayed Value attribute of the dimension is OFF, the number of decimal places of the dimension is modified and the stored value is rounded to the specified number of decimal places.

  For example, consider a dimension with its stored value as 12.12323 and the Round Displayed Value attribute of the dimension is OFF. If the function `ProDimensionDecimalsSet()` sets the dimension to 3 decimal places, then the stored value of the dimension is rounded to 3 decimal places and is modified to 12.123. The dimension is displayed as 12.123.
- If the number of decimal places required to display the stored value of the dimension is less than the number of decimal places specified in the function `ProDimensionDecimalsSet()`, the number of decimal places is set to the specified value. The status of the Round Displayed Value attribute is not considered, as no change or an increase to the number of decimal places will have no effect on the stored value.

For example, consider a dimension with its stored value as 12.12323. If the function `ProDimensionDecimalsSet()` sets the dimension to 8 decimal places and if trailing zeros are displayed, then the dimension is displayed as 12.12323000.

For a driven dimension:

- If the number of decimal places set by the function is greater than or equal to the number of decimal places required to display the stored value of the dimension, the decimal places value is changed and no change to the Round Displayed Value attribute is made.
- If the number of decimal places of the dimension is less than the number required to display the stored value of the dimension, the Round Displayed Value attribute is AUTOMATICALLY set to ON as it is not possible to change the stored value of a driven dimension.

Note

The value given for the input argument decimals of the function `ProDimensionDecimalsSet()` should be a non-negative number. It should be such that when you apply either the upper or the lower values of the tolerance to the given dimension, the total number of digits before and after the decimal point in the resulting values does not exceed 13.

The function `ProDimensionDenominatorSet()` sets the denominator for the fractional dimensions. When you call the function `ProDimensionDenominatorSet()`:

- The stored value remains unchanged if,
  - it can be expressed as an exact fraction with the given denominator, regardless of whether the round-off attribute is set or not.
  - the stored value cannot be expressed as an exact fraction, but the round-off attribute is set. In this case, the fraction is the approximate representation of the stored value.
- The stored value changes to the nearest fraction and triggers a regeneration of the model, if it cannot be expressed as an exact fraction with the given denominator and the round-off attribute is not set.

The functions `ProDimensionBoundSet()` sets the bound status of the dimension.

The function `ProDimensionDimensionReset()` sets the dimension to the value it had at the end of the last successful regeneration. You can use this function to recover from a failed regeneration.

The function `ProDimensionBasicSet()` and the function `ProDimensionInspectionSet()` set the basic and inspection notations of the dimension. These functions are applicable to both driven and driving dimensions.

Note

The basic and inspection notations of the dimension are not available when only the tolerance value for a dimension is displayed.

The function `ProDimensionElbowSet()` sets the length of the elbow for the specified dimension in a solid. The function can also be used to set the length of the elbow for a dimension in a drawing, where the dimension is created in a solid and is displayed in a drawing. To work with dimensions shown in a drawing, pass the name of the drawing in the input argument *drawing*.

The function `ProDimensionSimpleBreakCreate()` creates a simple break on an existing dimension witness line. The input arguments are:

- `dimension`—Specifies a pointer to the dimension whose witness line is to be broken.
- `drawing`—Specifies the drawing in which the dimension is present. You can specify a `NULL` value.
- `index`—Specifies the index of the witness line. Depending on which side of the dimension the witness line lies, specify the value as `1` or `2`. Use the method `ProDimlocationWitnesslinesGet` to get the location of the witness line end points for a dimension.

Note

This argument is not applicable for ordinate, radius, and diameter dimensions.

- `break_start`—Specifies the start point of the break.
- `break_end`—Specifies the end point of the break.

The function `ProDimensionJogCreate()` creates a jog on an existing dimension witness line. The input arguments are:

- `dimension`—Specifies a pointer to the dimension where the jog will be created.
- `drawing`—Specifies the drawing in which the dimension is present. You can specify a `NULL` value.

- `index`—Specifies the index of the witness line. Depending on which side of the dimension the witness line lies, specify the value as `1` or `2`. Use the method `ProDimlocationWitnesslinesGet` to get the location of the witness line end points for a dimension.

  Note

  This argument is not applicable for ordinate, radius, and diameter dimensions.

- `jog_points`—Specifies an array of points to position the jog. If the specified witness line has no jog added to it, then you must specify minimum two points that is, the start point and end point of the jog.

Note

The functions `ProDimensionSimpleBreakCreate()` and `ProDimensionJogCreate()` return the error type `PRO_TK_INVALID_TYPE` when breaks and jogs are not supported for the specified dimension type, for example, diameter dimension.

The functions return the error type `PRO_TK_ABORT` when it is not possible to create breaks or jogs for the specified dimension witness line. For example, if you add a jog that is duplicate to an existing jog on the dimension witness line.

The function `ProDimensionArrowtypeSet()` sets the arrow type for the specified arrow in a dimension. The input arguments are:

- *dimension*—Specifies the dimension.
- *drawing*—Specifies the drawing in which the dimension is displayed. To set the arrow type in the owner model, specify the argument value as `NULL`.
- *arrow_index*—Specifies the index of the witness line. Depending on which side of the dimension the witness line lies, specify the value as `1` or `2`.

  Note

  The value of *arrow_index* is ignored for ordinate and radius dimensions.

- *arrow_type*—Specifies the type of arrow using the enumerated data type `ProLeaderType`.

The function `ProDimensionWitnesslineErase()` erases a specified witness line from the dimension. The input arguments are:

- *dimension*—Specifies the dimension whose witness line must be erased. This argument cannot be `NULL`.
- *drawing*—Specifies the drawing in which the dimension is displayed. To erase a witness line from a solid, specify this argument as `NULL`.
- *WitnesslineIndex*—Specifies the index of the witness line. Specify the value as `1` or `2` depending on which side of the dimension the witness line lies. Use the method `ProDimlocationWitnesslinesGet()` to get the location of the witness line end points for a dimension.

Use the function `ProDimensionWitnesslineShow()` to show the erased witness line for the specified dimension.

Note

The functions `ProDimensionWitnesslineErase()` and `ProDimensionWitnesslineShow()` erase and show the witness lines of dimensions and reference dimensions, respectively. These functions work with both drawings and solids.

The function `ProDimensionDisplayFormatSet()` sets the format in which the specified dimension must be displayed.

Use the function `ProDimensionOriginSideSet()` to set a witness line as the origin or start of measurement for the specified dimension. Specify the index of the witness line in the input argument *dim_side*. If a witness line is already set as origin, pass *dim_side* as `-1` to remove the origin.

The function `ProDimensionEnvelopeGet()` returns the envelope of a line in the specified dimension. While retrieving coordinates of the dimension in a specified solid, if the dimension is displayed in the solid as well as in the drawing, the drawing must not be active. The input arguments follow:

- *dimension*—Dimension
- *drawing*—Drawing. The value for this input argument must be passed only if the solid dimension is shown in the drawing. Else, pass it as `NULL`.
- *line_number*—The line number of the dimension. To get a full dimension envelope, pass this value as `PRO_VALUE_UNUSED`.

The output argument `envelope` is the envelope surrounding the text line in the model coordinate system. For drawing, the envelope surrounding the dimension is in the screen coordinates.

Example 3: Modifying a Dimension

The sample code in the file `UgDimsChange.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dims` demonstrates how to modify a dimension. The dimensions belonging to a given feature are displayed. You can choose the dimension you wish to modify.

Dimension References

The functions explained in this section enable you to work with semantic dimension references.

Function Introduced:

- **ProDimensionAdditionalRefsAdd()**
- **ProDimensionAdditionalRefsGet()**
- **ProDimensionAdditionalRefDelete()**

The function `ProDimensionAdditionalRefsAdd()` adds additional semantic references in the specified dimension. The input arguments are:

- *dim*—Specifies a dimension.
- *type*—Specifies the type of reference using the enumerated data type `ProDimensionReferenceType`. The type is classified based on the list to which the references are added. The valid values are:
  - `PRO_DIM_REF_FIRST`—Adds the semantic references to the first list of references.
  - `PRO_DIM_REF_SECOND`—Adds the semantic references to the second list of references.

- PRO_DIM_SRF_COLL—Adds the semantic references to the collection of surfaces.

Note

When a reference includes more than one collection, the function ProDimensionAdditionalRefsAdd() returns the error PRO_TK_MAX_LIMIT_REACHED and no reference is added.

- *refs* —Specifies a ProArray of references that will be added to the specified dimension.

Note

Currently, the reference types PRO_ANNOT_REF_SINGLE and PRO_ANNOT_REF_SRF_COLLECTION are supported.

The function ProDimensionAdditionalRefsGet() returns a ProArray of references of the specified type for a dimension. In the input argument *type*, specify the type of reference using the enumerated data type ProDimensionReferenceType. Use the function ProAnnotationreferencearrayFree() to release the memory assigned to the ProArray of references.

Use the function ProDimensionAdditionalRefDelete() to delete the specified reference. The references are specified by their index number which start from 0. You can get existing references from ProDimensionAdditionalRefsGet(). The index is ignored if the type of reference is surface collection, as only one reference of the type PRO_DIM_SRF_COLL can exist.

The input argument for both the functions ProDimensionAdditionalRefsGet() and ProDimensionAdditionalRefDelete() specifies the type of reference using the enumerated data type ProDimensionReferenceType. The type is classified based on the list to which the references are added. The valid values are:

- PRO_DIM_REF_FIRST—Adds the semantic references to the first list of references.
- PRO_DIM_REF_SECOND—Adds the semantic references to the second list of references.
- PRO_DIM_SRF_COLL—Adds the semantic references to the collection of surfaces.

Clean Up Dimensions

You can clean up the placement of dimensions in a drawing to meet the industry standards, and enable easier reading of your model detailing. You can adjust the location and display of dimensions by setting controls on the placement of a dimension. You can also set the cosmetic attributes, like flip the direction of arrow when the arrows do not fit between the witness lines and center the dimension text between two witness lines.

Function Introduced:

- **ProDrawingDimensionsCleanup()**

Use the function `ProDrawingDimensionsCleanup()` to clean up the dimensions in a drawing. The input arguments are:

- *draw*—Specifies the drawing.
- *view*—Specifies the view in which the dimensions must be cleaned. If you pass the value as `NULL`, the dimensions are cleaned for all the views in the specified drawing.

The dimensions are cleaned using the default values set in the **Clean Dimensions** dialog box in Creo Parametric user interface.

Dimension Tolerances

Functions Introduced:

- **ProToleranceDefaultGet()**
- **ProDimensionDisplayedToleranceGet()**
- **ProSolidToleranceGet()**
- **ProSolidToleranceSet()**
- **ProDimensionIsToleranceDisplayed()**
- **ProDimensionToltypeGet()**
- **ProDimensionToltypeSet()**
- **ProDimensionToleranceGet()**
- **ProDimensionToleranceSet()**
- **ProDimensionTolerancedecimalsGet()**
- **ProDimensionTolerancedecimalsSet()**
- **ProDimensionTolerancedenominatorGet()**
- **ProDimensionTolerancedenominatorSet()**

The function ProToleranceDefaultGet() tells you the current default value for a given tolerance, that is, the value set in the configuration file to be used for new models. There is a separate tolerance for linear and angular dimensions (PROTOLERANCE_LINEAR and PROTOLERANCE_ANGULAR) and for each number of decimal places in the range 1 to 6 (12 tolerance settings in all).

If the round off attribute for the given dimension is set, the function ProDimensionDisplayedToleranceGet() retrieves the displayed rounded values of the upper and lower limits of the specified dimension. Otherwise, it retrieves the stored values of the tolerances as done by the function ProSolidToleranceGet(). For example, consider a dimension that is set to round off to two decimal places and has the upper and lower tolerances 0.123456. By default, the tolerance values displayed are also rounded off to two decimal places. In this case, the function ProDimensionDisplayedToleranceGet() retrieves the upper and lower values as 0.12.

The input argument of the function ProDimensionDisplayedToleranceGet() is a dimension handle. The output arguments of this function are pointers to the rounded values of the upper and lower limits of the specified dimension. You must allocate a memory location for each of the output arguments. Pass a NULL pointer if you do not want to use an output argument. You cannot pass a null for both the output arguments.

The functions ProSolidToleranceGet() and ProSolidToleranceSet() let you find and set the current value of the specified dimensional tolerance. The function ProDimensionIsToleranceDisplayed() tells you whether the tolerances of the specified dimension are currently being displayed. Refer to the Creo Parametric Detailed Drawings Help for more information.

The function ProDimensionToltypeGet() returns the display format for the specified dimension tolerance using the enumerated data type ProDimToleranceType. The valid values are:

- PRO_TOL_DEFAULT—Displays dimensions without tolerances. Similar to the nominal option in Creo Parametric.
- PRO_TOL_LIMITS—Displays dimension tolerances as upper and lower limits.

> **Note**
>
> This format is not available when only the tolerance value for a dimension is displayed.

- PRO_TOL_PLUS_MINUS—Displays dimensions as nominal with plus-minus tolerances. The positive and negative values are independent.

- PRO_TOL_PLUS_MINUS_SYM—Displays dimensions as nominal with a single value for both the positive and the negative tolerance.
- PRO_DIM_TOL_SYM_SUPERSCRIPT—Displays dimensions as nominal with a single value for positive and negative tolerance. The text of the tolerance is displayed in a superscript format with respect to the dimension text.
- PRO_DIM_TOL_BASIC—Displays dimensions as basic dimensions. Basic dimensions are displayed in an enclosed feature control frame . Tolerances are not displayed in basic dimensions, only the numerical part of the dimension value and its symbol are enclosed in the rectangular box. Any additional text in the dimension value is not included in the box.

Use the function `ProDimensionToltypeSet()` to set the display format for the specified dimension tolerance.

The function `ProDimensionToleranceGet()` reports the deviation of the upper and lower tolerances from the nominal value. The values are always reported as independent upper and lower tolerance values; the actual display of the tolerance is determined by `ProDimensionToltypeGet()`. Tolerances are not applicable for reference dimensions.

Note

If the upper tolerance value is negative, it will be displayed with a '-' sign. But if the lower tolerance value is negative, it will be displayed with a '+' sign.

The function `ProDimensionToleranceSet()` sets the upper and lower tolerance limits. The values are always accepted as independent upper and lower tolerance values; the actual display of the tolerance is determined by `ProDimensionToltypeSet()`. It is not applicable to reference dimensions.

The functions `ProDimensionTolerancedecimalsGet()` and `ProDimensionTolerancedecimalsSet()` obtain and assign the number of decimal places shown for the upper and lower values of the dimension tolerance. Thus, the decimals of the dimension tolerance can be set independent of the number of dimension decimals. By default, the number of decimal places for tolerance values is calculated based upon the "linear_tol" settings of the model.

Note

The input *tolerance_decimals* to the `ProDimensionTolerancedecimalsSet()` function should be a non-negative number and it should be such that when you apply either the upper or the lower values of the tolerance to the given dimension, the total number of digits before and after the decimal point in the resulting values does not exceed 13.

If the dimension tolerance value is a fraction, the functions `ProDimensionTolerancedenominatorGet()` and `ProDimensiondenominatorSet()` obtain and assign the value for the largest possible denominator for the upper and lower tolerance values. By default, this value is defined by the `config.pro` option, `dim_fraction_denominator`.

ISO/DIN Tolerance Table Use

Functions Introduced:

- **ProSolidModelclassGet()**
- **ProSolidModelclassSet()**
- **ProSolidTolclassLoad()**
- **ProDimensionTollabelGet()**
- **ProDimensionTollabelSet()**

Creo Parametric TOOLKIT provides functions that programmatically set and return `ISO/DIN` tolerance table data. These functions allow changes to values before the label is set. For all other labels, use the `ProDimensionTollabelSet()` command.

The functions `ProSolidModelclassGet()` and `ProSolidModelclassSet()` respectively return or set the type of tolerance to use for a particular model. Valid settings are:

- `COARSE`
- `FINE`
- `MEDIUM`
- `VERY_COARSE`

The function `ProSolidTolclassLoad()` loads a hole or shaft `ISO/DIN` tolerance table into the current session memory.

The functions `ProDimensionTollabelGet()` and `ProDimensionTollabelSet()` respectively get or set the `ISO/DIN` tolerance table assigned to the specified dimension.

Dimension Text

Functions Introduced:

- **ProDimensionTextWstringsGet()**
- **ProDimensionTextWstringsSet()**

Superseded Functions:

- **ProDimensionTextGet()**
- **ProDimensionTextSet()**

The functions `ProDimensionTextGet()` and `ProDimensionTextSet()` have been deprecated. Use the functions `ProDimensionTextWstringsGet()` and `ProDimensionTextWstringsSet()` instead.

The function `ProDimensionTextWstringsGet()` retrieves the text of the specified dimension as a `ProArray` of wide character strings. Use the function `ProWstringproarrayFree()` to release the memory allocated for the `ProArray`.

The function `ProDimensionTextWstringsSet()` sets the text of the specified dimension using a `ProArray` of wide character strings. This is equivalent to editing dimensions in Creo Parametric.

Note

From Creo Parametric 2.0 onward, the functions `ProDimensionTextWstringsGet()` and `ProDimensionTextGet()` will always include the dimension value `@D` that appears in the **Dimension Properties** dialog box.

Dimension Text Style

Functions Introduced:

- **ProDimensionTextstyleGet()**
- **ProDimensionTextstyleSet()**

The function `ProDimensionTextstyleGet()` returns the text style assigned to a specified dimension or reference dimension.

The function `ProDimensionTextstyleSet()` sets the text style assigned to a specified dimension or reference dimension.

Note

Only some of the text style properties may be assigned to dimensions.

Dimension Prefix and Suffix

Functions Introduced:

- **ProDimensionPrefixGet()**
- **ProDimensionPrefixSet()**
- **ProDimensionSuffixGet()**
- **ProDimensionSuffixSet()**

The function `ProDimensionPrefixGet()` retrieves the prefix assigned to a specified dimension. Use the function `ProDimensionPrefixSet()` to set the prefix of the type `ProLine` for a dimension.

The function `ProDimensionSuffixGet()` retrieves the suffix assigned to a specified dimension. Use the function `ProDimensionSuffixSet()` to set the suffix of the type `ProLine` for a dimension.

Dimension Location

The functions described in this section extract the dimension location and geometry in 3D space for solid model dimensions.

Functions Introduced:

- **ProDimensionLocationGet()**
- **ProDimlocationFree()**
- **ProDimensionMove()**

The function `ProDimensionLocationGet()` returns the location of the elements that make up a solid dimension or reference dimension.

This function optionally takes a view used to determine the orientation of the model when calculating the dimension locations. The orientation often determines the text location, presence or absence of elbows, and other dimension location properties.

Use the function `ProDimlocationFree()` to free the structure containing the dimension location data.

The function `ProDimensionMove()` enables you to move the specified dimension to the given location within its owner model.

Dimension Entity Location

The following functions extract the locations of geometric endpoints for the dimension. You can calculate the dimension location plane, witness line, and dimension orientation vectors from these points. The location of the points is specified in the same coordinate system as the solid model.

Functions Introduced:

- **ProDimlocationTextGet()**
- **ProDimlocationArrowsGet()**
- **ProDimlocationWitnesslinesGet()**
- **ProDimlocationArrowtypesGet()**
- **ProDimlocationCenterleadertypeGet()**
- **ProDimlocationZExtensionlinesGet()**
- **ProDimlocationNormalGet()**

The function `ProDimlocationTextGet()` returns the location of the dimension text in model coordinates. If the dimension contains an elbow, the function returns the location of the elbow joint and the length of the elbow in model coordinates.

Note

If the value of the elbow length was originally set by the function `ProDimensionElbowlengthSet()`, then the value returned by `ProDimlocationTextGet()` is equal to the value set by the function `ProDimensionElbowlengthSet()` minus the padding around the text for all dimension types.

The function `ProDimlocationArrowsGet()` returns the location of the arrow heads for the dimension.

The function `ProDimlocationWitnesslinesGet()` returns the location of the witness line ends for the dimension.

The function `ProDimlocationArrowtypesGet()` returns the type of arrows used for the leader of a specified 3D dimension. The dimension location obtained using the function `ProDimensionLocationGet()` serves as an input argument for this function.

Note

In case of radial dimensions where the arrow head type cannot be changed, the function `ProDimlocationArrowtypesGet()` always returns the "Arrow head" leader type.

The function `ProDimlocationCenterleadertypeGet()` obtains the type of center leader used for the dimension, if the dimension uses a center leader. The type of center leader is determined by the orientation of the dimension text. This function also returns the length and direction of the elbow used by the center leader and the leader end symbol.

- `PRO_DIM_CLEADER_CENTERED_ELBOW`—Specifies that the dimension text is placed next to and centered about the elbow of the center leader.
- `PRO_DIM_CLEADER_ABOVE_ELBOW`—Specifies that the dimension text is placed next to and above the elbow of the center leader.
- `PRO_DIM_CLEADER_ABOVE_EXT_ELBOW`—Specifies that the dimension text is placed above the extended elbow of the center leader.
- `PRO_DIM_PARALLEL_ABOVE`—Specifies that the dimension text is placed parallel to and above the center leader.
- `PRO_DIM_PARALLEL_BELOW`—Specifies that the dimension text is placed parallel to and below the center leader.

Note

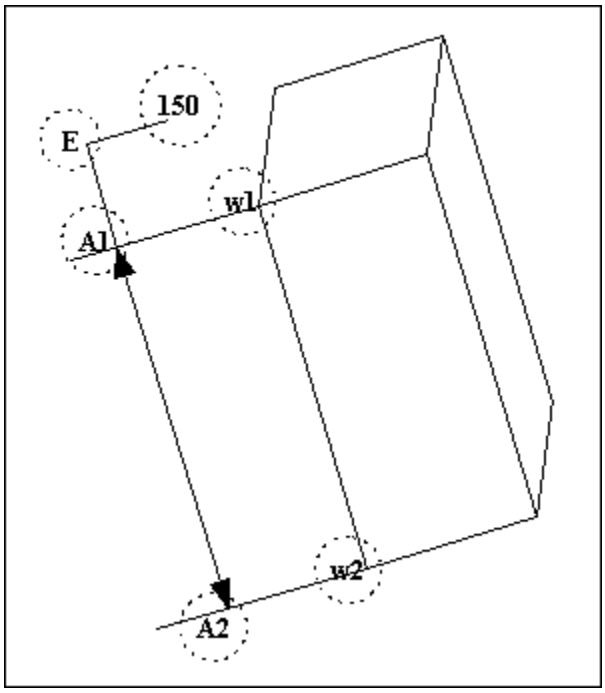
- A center leader type is available only for linear and diameter dimensions.
- The elbow length and direction is not available for `PRO_DIM_PARALLEL_ABOVE` and `PRO_DIM_PARALLEL_BELOW` center leader types.

The function `ProDimlocationZExtensionlinesGet()` obtains the endpoints of the Z-extension line created for a specified dimension. Z-extension lines are automatically created whenever the dimension's attachment does not intersect its reference in the Z-Direction. The Z-extension line is attached at the edge of the surface at the closest distance from the dimension witness line.

The function `ProDimlocationNormalGet()` returns the vector normal to the dimensioning plane for a radial or diameter dimension. This normal vector should correspond to the axis normal to the arc being measured by the radial or diameter dimension.

The following figures illustrate the potential location of the arrow heads and witness lines for different dimension types.

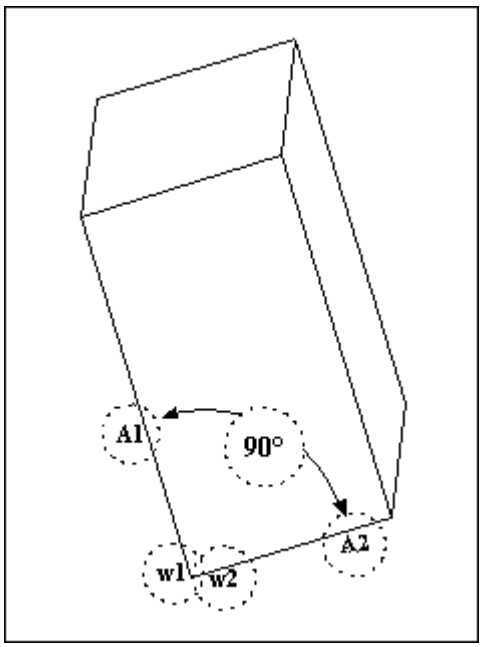
Linear and psuedo-linear dimensions



For a linear type of dimension, there are typically two arrow locations A1 and A2 as shown in the above figure. w1 and w2 indicate the two witness line locations.

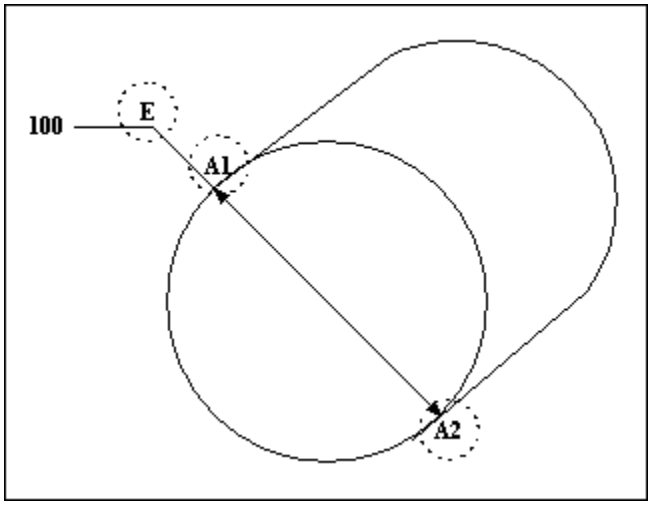
If the dimension type has an elbow joint indicated by E, the elbow length is the distance between the text and E. If the dimension does not have an elbow, the text occurs on the line between A1 and A2, and its position is returned by the function `ProDimensionTextGet()`. Pattern parameter dimensions and length-of-arc dimensions also typically return this dimension structure.

Angular dimensions



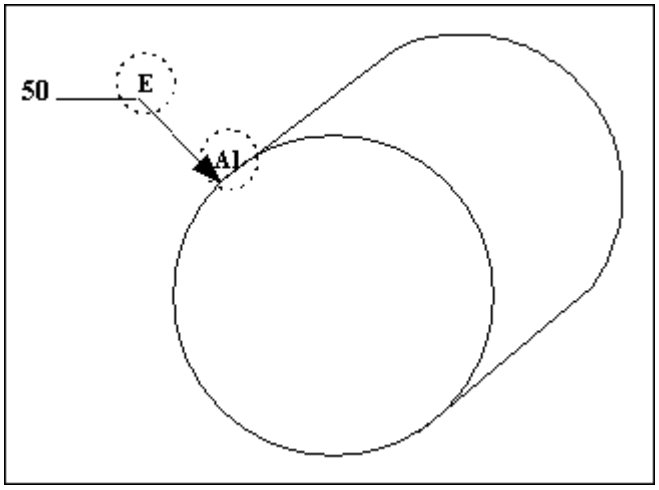
For an angular type of dimension, there are two arrow locations A1 and A2 as shown in the above figure. w1 and w2 indicate the two witness line locations. For some angular dimensions the two witness line endpoints are coincident, but they are returned as independent locations. This dimension type does not have an elbow joint.

Diameter dimensions



For a diameter type of dimension, there are two arrow locations A1 and A2 as shown in the above figure. The elbow joint for this dimension is indicated by E. The elbow length is the distance between the text and the elbow joint. This dimension type does not have any witness line locations.

Radius dimensions



For a radius type dimension, there is one arrow location indicated by A1 and an elbow joint indicated by E. The elbow length is the distance between the text and the elbow joint.

The function `ProDimlocationArrowsGet()` returns a `NULL` value for the second arrow location. This dimension type does not have any witness line locations.

Example 4: Dimension Location Properties

The sample code in the file `UgDimLocationUtils.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dims` demonstrates use of the dimension location properties. It finds linear dimensions that are oriented parallel to a specified direction vector. It shows these dimensions in the Creo Parametric user interface. The orientation of the dimension is determined from the coordinates of the two dimension arrowheads.

Dimension Orientation

Functions Introduced:

- **ProDimensionPlaneSet()**
- **ProDimensionPlaneGet()**

The function `ProDimensionPlaneSet()` assigns an annotation plane as the orientation of a specified dimension stored in an annotation element.

The function `ProDimensionPlaneGet()` obtains the orientation of a specified dimension stored in an annotation element.

Note

For dimensions that are not assigned a specific orientation, the orientation obtained includes `PRO_ID_NO_ANNOTATION_PLANE` as its ID.

Driving Dimension Annotation Elements

You can convert driving dimensions created by features into annotation elements and place them on annotation planes. However, you can create the driving dimension annotation elements only in the features that own the dimensions. These annotation elements cannot have any user defined or system references.

In Creo Parametric TOOLKIT, a driving dimension annotation element uses the `ProModelitem` handle with the type field `PRO_DIMENSION` and the appropriate dimension ID.

Functions such as `ProSolidDimensionVisit()` and `ProFeatureDimensionVisit()` can be used to find both edit dimensions and driving dimension annotation elements.

Functions Introduced:

- **ProDimensionAnnotationelemCreate()**
- **ProDimensionAnnotationelemDelete()**

The function `ProDimensionAnnotationelemCreate()` creates an annotation element for a specified driving dimension, based on the desired annotation orientation.

The function `ProDimensionAnnotationelemDelete()` removes the annotation element containing the driving dimension. It deletes all the parameters and relations associated with the annotation element.

Accessing Reference and Driven Dimensions

The functions described in this section provide additional access to reference and driven dimension annotations.

Many functions listed in the previous sections that are applicable for driving dimensions are also applicable for reference and driven dimensions.

Functions Introduced:

- **ProDimensionIsDriving()**
- **ProDimensionCreate()**
- **ProDimensionAttachmentsGet()**

- **ProDimensionAttachmentsSet()**
- **ProDimensionDelete()**
- **ProDimensionCanRegenerate()**
- **ProDimensionCanRegen()**

The function `ProDimensionIsDriving()` determines if a dimension is driving geometry or is driven by it. If a dimension drives geometry, its value can be modified and the model regenerated with the given change. If a dimension is driven by geometry, its value is fixed but it can be deleted and redefined as necessary. A driven dimension may also be included in an annotation element.

The function `ProDimensionCreate()` creates a new driven or reference dimension. Specify the geometric references and parameters required to construct the required dimension as the input parameters for this function. Once the reference dimension is created, use the function `ProAnnotationShow()` to display it. The input arguments of this function are as follows:

- *model*—Specifies the solid model.
- *dimension_type*—Specifies the type of dimension. This parameter can have the following values:
  - `PRO_REF_DIMENSION`—Specifies a reference dimension.
  - `PRO_DIMENSION`—Specifies a driven dimension.
- *annotation_plane*—Specifies the annotation plane for the dimensions.
- *attachments_arr*—Specifies the points on the model where you want to attach the dimension.

> **Note**
>
> The attachments structure is an array of two `ProSelection` entities. It is provided to support options such as intersect where two entities must be passed as input. From Creo Parametric 3.0 onward, you can create dimensions that have intersection type of reference. The intersection type of reference is a reference that is derived from the intersection of two entities. Refer to the Creo Parametric Detailed Drawings Help for more information on intersection type of reference.

- *dsense_arr*—Specifies more information about how the dimension attaches to each attachment point of the model, that is, to what part of the entity
- *orient_hint*—Specifies the orientation of the dimension and has one of the following values:
  - `PRO_DIM_ORNT_HORIZ`—Specifies a horizontal dimension.

- PRO_DIM_ORNT_VERT—Specifies a vertical dimension.
- PRO_DIM_ORNT_SLANTED—Specifies the shortest distance between two attachment points (available only when the dimension is attached to points).
- PRO_DIM_ORNT_ELPS_RAD1—Specifies the start radius for a dimension on an ellipse.
- PRO_DIM_ORNT_ELPS_RAD2—Specifies the end radius for a dimension on an ellipse.
- PRO_DIM_ORNT_ARC_ANG—Specifies the angle of the arc for a dimension of an arc.
- PRO_DIM_ORNT_ARC_LENGTH—Specifies the length of the arc for a dimension of an arc.
- PRO_DIM_ORNT_LIN_TANCRV_ANG—Specifies the value to dimension the angle between the line and the tangent at a curve point (the point on the curve must be an endpoint).
- PRO_DIM_ORNT_RAD_DIFF—Specifies the linear dimension of the radial distance between two concentric arcs or circles.
- PRO_DIM_ORNT_NORMAL—Specifies the linear dimension between two points to be placed normal to the selected reference.
- PRO_DIM_ORNT_PARALLEL—Specifies the linear dimension between two points to be placed parallel to the selected reference.

- *location*—Specifies the initial location of the dimension text.

The function ProDimensionAttachmentsGet() and ProDimensionAttachmentSet() provide access to the geometric references and parameters of the driven or reference dimension. These functions support dimensions that are created with intersection type of reference.

The function ProDimensionDelete() deletes the driven or reference dimension. Dimensions stored in annotation elements should be deleted using ProAnnotationfeatElementDelete().

The function ProDimensionCanRegenerate() supersedes the function ProDimensionCanRegen(). The function ProDimensionCanRegenerate() checks if a driven dimension can be regenerated. For driven dimensions created in drawing mode and owned by a solid, which can be displayed only in the context of that drawing, specify the name of the drawing in the input argument *drawing*. The function ProDimensionCanRegen() is equivalent to ProDimensionCanRegenerate() when the input argument *drawing* is NULL.

45-degree Chamfer Dimensions

You can create 45-degree chamfer dimensions by referencing one of the following items:

- Edges, including solid or surface edges, silhouette edges, curves, and sketches.
- Surfaces
- Revolve surfaces

The functions described in this section provide access to the display style of 45-degree chamfer dimensions in a solid. These functions can also be used to access the display style of the chamfer dimension in a drawing, where the dimension is created in a solid and is displayed in a drawing. To work with dimensions shown in a drawing, pass the name of the drawing in the input argument *drawing* in the functions.

Note

The default display of a 45-degree chamfer dimension depends upon the setting of the `config.pro` option, `default_chamfer_text`.

Functions Introduced:

- **ProDimensionChamferLeaderGet()**
- **ProDimensionChamferLeaderSet()**
- **ProDimensionChamferTextGet()**
- **ProDimensionChamferTextSet()**
- **ProDimensionConfigGet()**
- **ProDimensionConfigSet()**

From Creo Parametric 3.0 M060 onward, the functions `ProDimensionChamferLdrstyleGet()` and `ProDimensionChamferLdrstyleSet()` have been deprecated. Use the functions `ProDimensionChamferLeaderGet()` and `ProDimensionChamferLeaderSet()` instead.

The functions `ProDimensionChamferLeaderGet()` and `ProDimensionChamferLeaderSet()` retrieve and set the style of the leader for the specified 45-degree chamfer dimension. The valid values are as follows:

- PRO_DIM_CHMFR_LEADER_STYLE_NORMAL—Specifies that the leader of a chamfer dimension is normal to the chamfer edge (ASME, ANSI, JIS, ISO Standard).
- PRO_DIM_CHMFR_LEADER_STYLE_LINEAR—Specifies that the leader of a chamfer dimension has linear style of display.
- PRO_DIM_CHMFR_LEADER_STYLE_DEFAULT—Specifies that the chamfer dimension leader style should be displayed using the default value set in the detail option default_chamfer_leader_style.

From Creo Parametric 3.0 M060 onward, the functions ProDimensionChamferstyleGet() and ProDimensionChamferstyleSet() have been deprecated. Use the functions ProDimensionChamferTextGet() and ProDimensionChamferTextSet() instead.

The functions ProDimensionChamferTextGet() and ProDimensionChamferTextSet() retrieve and set the dimension scheme for the specified 45-degree chamfer dimension. The valid values are as follows:

- PRO_DIM_CHMFRSTYLE_CD—Specifies that the chamfer dimension text should be displayed in the C(Dimension value) format (JIS/GB Standard).
- PRO_DIM_CHMFRSTYLE_D_X_45—Specifies that the chamfer dimension text should be displayed in the (Dimension value) X 45 format (ISO/DIN Standards).
- PRO_DIM_CHMFRSTYLE_DEFAULT—Specifies that the chamfer dimension text should be displayed using the default value set in the drawing detail optiondefault_chamfer_text.
- PRO_DIM_CHMFRSTYLE_45_X_D—Specifies that the chamfer dimension text should be displayed in the 45 X (Dimension value) format (ASME/ANSI Standards).

From Creo Parametric 3.0 M060 onward, the functions ProDimensionConfigurationGet() and ProDimensionConfigurationSet() have been deprecated. Use the functions ProDimensionConfigGet() and ProDimensionConfigSet() instead.

The functions ProDimensionConfigGet() and ProDimensionConfigSet() retrieve and set the dimension configuration for chamfer dimensions. The dimension configuration defines the style in which the dimension must be displayed. The valid values are as follows:

- `PRO_DIMCONFIG_LEADER`—Creates the dimension with a leader.
- `PRO_DIMCONFIG_LINEAR`—Creates a linear dimension.
- `PRO_DIMCONFIG_CENTER_LEADER`—Creates the dimension with the leader note attached to the center of the dimension leader line.

Accessing Ordinate and Baseline Dimensions

The functions described in this section enable you to create 3D ordinate driven dimensions in 3D models as model annotations or as annotation elements. They also provide the ability to define a baseline annotation element, and then define model ordinate dimension annotations and ordinate dimension annotation elements that reference the baseline annotation element.

Baseline Dimensions

Functions Introduced:

- **ProAnnotationfeatBaselineCreate()**
- **ProDimensionIsBaseline()**

The function `ProAnnotationfeatBaselineCreate()` creates an ordinate baseline annotation element and corresponding dimension. Specify the feature reference geometry, text location, direction and annotation plane as input arguments for this function.

The function `ProDimensionIsBaseline()` identifies whether a dimension is a baseline dimension.

Ordinate Dimensions

Function Introduced:

- **ProDimensionOrdinateCreate()**
- **ProDimensionOrdinatestandardGet()**
- **ProDimensionOrdinatestandardSet()**
- **ProDimensionOrdinatereferencesSet()**
- **ProDimensionMove()**
- **ProDimensionIsOrdinate()**
- **ProDimensionAutoOrdinateCreate()**

The function `ProDimensionOrdinateCreate()` creates a new model ordinate driven dimension or a model ordinate reference dimension in a solid model. It requires the input of a reference baseline annotation as well as a

geometry reference. The annotation plane for the new dimension will be inherited from the baseline. Once the reference dimension is created, use the function `ProAnnotationShow()` to display it.

To create an ordinate driven dimension element or a model ordinate reference dimension pass the ordinate dimensions created by `ProDimensionOrdinateCreate()` to the function `ProAnnotationelemAnnotationSet()`.

The function `ProDimensionOrdinatestandardGet()` returns the display standard for the ordinate dimensions in the drawing. The style of the ordinate dimension may be as follows:

- `PRO_DIM_ORDSTD_DEFAULT`—Specifies the default style for the ordinate dimensions.
- `PRO_DIM_ORDSTD_ANSI`—Specifies the American National Standard style for the ordinate dimension. It places the related ordinate dimensions without a connecting line.
- `PRO_DIM_ORDSTD_JIS`—Specifies the Japanese Industrial Standard style for the ordinate dimension. It places the ordinate dimensions along a connecting line that is perpendicular to the baseline and starts with an open circle.
- `PRO_DIM_ORDSTD_ISO`—Specifies the International Standard of Organization style for the ordinate dimension.
- `PRO_DIM_ORDSTD_DIN`—Specifies the German Institute for Standardization style for the ordinate dimension.
- `PRO_DIM_ORDSTD_SAME_AS_3D`—Specifies the ordinate dimension style for 2D drawings. Not used in 3D ordinate dimensions.

The function `ProDimensionOrdinatestandardSet()` sets the style for the specified ordinate dimension or a set of ordinate dimensions.

The function `ProDimensionAttachmentsGet()` returns the attachment geometry for the dimension.

In order to change the dimension's attachments you must use the function `ProDimensionOrdinatereferencesSet()`.

The function `ProDimensionMove()` enables you to move the specified 3D ordinate dimension to the specified location within its owner model. For ordinate dimensions in the `JIS` or `ISO/DIN` style, all dimensions stay aligned during movement. For ordinate dimensions in the `ANSI` style, each dimension can be adjusted independent of the other dimensions. If the style is changed back to `JIS` or `ISO/DIN`, all the dimensions become aligned with the baseline.

The function `ProDimensionIsOrdinate()` identifies if a dimension is ordinate.

The function ProDimensionAutoOrdinateCreate() creates ordinate dimensions automatically for the selected surfaces. The function returns a ProArray of dimensions. The input arguments are:

- drawing—Specifies the drawing where the ordinate dimensions must be automatically created.
- surface_array—Specifies a set of parallel surfaces for which the ordinate dimensions must be created. This is a ProArray of selection handles. You can free this array using the function ProSelectionarrayFree().
- baseline—Specifies a reference element used to create the baseline dimension. The reference element can be an edge, a curve, or a datum plane.

Notes

The functions in this section enable you to access the notes created in Creo Parametric .

Note

These functions are applicable to solids (parts and assemblies) only. However, when notes on a solid are viewed from Drawing mode, they can also be accessed using the ProDtlnote() functions described in the chapter Drawings on page 1226.

A note is modeled in Creo Parametric TOOLKIT as an instance of ProModelitem with the type PRO_NOTE. You can select a note by supplying the selection option note_3d to ProSelect(). You can access the name of a note using the functions ProModelitemNameGet() and ProModelitemNameSet().

Creating and Deleting Notes

Functions Introduced:

- **ProSolidNoteCreate()**
- **ProNoteDelete()**

The function ProSolidNoteCreate() takes as input a ProMdl for the solid (either a part or an assembly), a ProModelitem for the owner of the note, and an expandable array ProLineList for the note text. The function outputs a ProNote object for the created note. Once the note is created, use the function ProAnnotationShow() to display it.

The function `ProNoteDelete()` deletes the note specified by its `ProNote` object. Notes stored in annotation elements should be deleted using `ProAnnotationfeatElementDelete()`.

Note Properties

Functions Introduced:

- **ProNoteTextGet()**
- **ProNoteTextSet()**
- **ProNoteURLGet()**
- **ProNoteURLSet()**
- **ProNoteURLWstringGet()**
- **ProNoteURLWstringSet()**
- **ProNoteURLExtraInfoGet()**
- **ProNoteURLExtraInfoSet()**
- **ProNoteOwnerGet()**
- **ProNoteLeaderstyleGet()**
- **ProNoteLeaderstyleSet()**
- **ProNoteElbowlengthGet()**
- **ProNoteElbowlengthSet()**
- **ProNoteLineEnvelopeGet()**
- **ProNoteAttachNormtanleaderGet()**
- **ProNoteAttachScreenSet()**
- **ProNoteWrapTextGet()**
- **ProNoteWrapTextSet()**
- **ProNoteReferencesAdd()**
- **ProNoteReferencesGet()**
- **ProNoteReferenceDelete()**

The function `ProNoteTextGet()` returns the text of a 3D model note. The function `ProNoteTextSet()` modifies the text of an exisitng 3D model note. You can also make symbols to be called out in the 3D notes using the function `ProNoteTextSet()`.

Use the function `ProNoteUndisplay()` followed by `ProNoteDisplay()` to update the display status of the note.

The function `ProNoteURLGet()` retrieves the Uniform Resource Locator (URL) associated with the specified note, whereas `ProNoteURLSet()` sets the associated URL for the specified note. Specify the note and the URL wide string or the name of the valid combined state as input arguments to the function `ProNoteURLWstringSet`.

Note

The functions `ProNoteURLGet()` and `ProNoteURLSet()` have been deprecated. Instead, use the functions `ProNoteURLWstringGet()` and `ProNoteURLWstringSet()` that return and set, respectively, the Uniform Resource Locator (URL) associated with the specified note as a widestring.

The function `ProNoteURLExtraInfoGet()` retrieves the information of whether opening the URL for a specified note appends the extra information `"?<model name>+<note id>"`.

The function `ProNoteURLExtraInfoSet()` sets whether opening the URL for a specified note should append the extra info `"?<model name>+<note id>"`.

The input argument *p_note_item* to both the functions is the note for which the extra information needs to be appended.

The function `ProNoteOwnerGet()` retrieves the owner of the specified note. The owner can be the solid model, a user-chosen feature, or the feature that contains the note's annotation element.

The function `ProNoteLeaderstyleGet()` returns the leader style used for the note. It can be either standard or ISO. The function `ProNoteLeaderstyleSet()` sets the leader style of the note.

The function `ProNoteElbowlengthGet()` returns the elbow properties for the specified note.

The function `ProNoteElbowlengthSet()` sets the elbow properties of the note. This is equivalent to **Move Text** option in the **Dimension Properties** dialog box in Creo Parametric.

Note

The elbow properties can be retrieved and set for the flat-to-screen notes and the drawing notes.

The function `ProNoteLineEnvelopeGet()` returns the envelope of a line for a specified note.

The function `ProNoteAttachNormtanleaderGet()` returns the properties of a leadered note which is normal/tangent to specified note.

Note

Creo Parametric adds hard line breaks to the multiple lines drawing notes created in Creo Elements/Pro during retrieval. The hard line breaks display the text of the note on separate lines in the **Note Properties** dialog box as they actually appear in the drawing note.

The function `ProNoteAttachScreenSet()` sets the location of the note text at the screen location. The input arguments follow:

- *note_attach*—Specifies the handle for `ProNoteAttach`.
- *p1*—The parameter in the X direction.
- *p2*—The parameter in the Y direction.
- *p3*—The parameter in the Z direction.

The functions `ProNoteWrapTextGet()` and `ProNoteWrapTextSet()` get and set the wrap status of the text for a specified note in a solid. The function `ProDtlnoteWrapTextSet()` sets the text wrapping status to ON or OFF.

The input arguments are listed below:

- *note*—Specifies the note for which the wrap status is to be set.
- *wrap*—Specifies if the text is wrapped. To wrap the text specify the value as `Pro_B_True`.
- *wrapwidth*—Specifies the width of the wrapped text line, if the input argument *wrap* is set to `Pro_B_True`.

The function `ProNoteReferencesAdd()` adds semantic references to a specified note in a solid. The input arguments are as follows:

- *note*—Specifies the note to which the additional semantic references are to be added.
- *refs*—Specifies the array of additional semantic references using the enumerated data type `ProAnnotationReference`.

Note

When a reference includes more than one collection, the function `ProNoteReferencesAdd()` returns the error `PRO_TK_MAX_LIMIT_REACHED` and no reference is added.

The function `ProNoteReferencesGet()` returns a `ProArray` of additional semantic references for a note.

The function `ProNoteReferenceDelete()` deletes the additional semantic references. The input arguments are as follows:

- *note* —Specifies the note from which the additional semantic references are to be deleted.
- *index_ref*—Specifies the index references. Indices start from 0.

Visiting Notes

Function introduced:

- **ProMdlNoteVisit()**

The function `ProMdlNoteVisit()` enables you to visit all the notes in the specified solid model.

Note Text Styles

Functions Introduced:

- **ProTextStyleAlloc()**
- **ProTextStyleFree()**
- **ProNoteTextStyleGet()**
- **ProNoteTextStyleSet()**

The function `ProTextStyleAlloc()` allocates the opaque handle for a `ProTextStyle` data structure. The function `ProTextStyleFree()` frees the allocated data structure.

The function `ProNoteTextStyleGet()` enables you to retrieve the text style of a specified note, whereas `ProNoteTextStyleSet()` enables you to set the text style of the note.

Text Style Properties

Functions introduced

- **ProTextStyleHeightGet()**
- **ProTextStyleHeightSet()**
- **ProTextStyleWidthGet()**
- **ProTextStyleWidthSet()**
- **ProTextStyleAngleGet()**
- **ProTextStyleAngleSet()**

- **ProTextStyleSlantAngleGet()**
- **ProTextStyleSlantAngleSet()**
- **ProTextStyleThicknessGet()**
- **ProTextStyleThicknessSet()**
- **ProTextStyleUnderlineGet()**
- **ProTextStyleUnderlineSet()**
- **ProTextStyleMirrorGet()**
- **ProTextStyleMirrorSet()**
- **ProTextStyleJustificationGet()**
- **ProTextStyleJustificationSet()**
- **ProTextStyleVertJustificationGet()**
- **ProTextStyleVertJustificationSet()**
- **ProTextStyleColorGetWithDef()**
- **ProTextStyleColorSetWithDef()**
- **ProTextStyleIsHeightInModelUnits()**
- **ProTextStyleHeightInModelUnitsSet()**
- **ProSolidDefaulttextheightGet()**

These functions enable you to retrieve and set the properties of the specified text style. You can retrieve and set text properties such as the height, width factor, angle, slant angle, thickness, underline, mirror.

Note

The system uses the value –1.0 for properties that use default values, or that have not been set yet.

The function `ProTextStyleJustificationGet()` returns the horizontal justification for the text style object.

The function `ProTextStyleJustificationSet()` sets the horizontal justification for the text style object using the enumerated data type `ProTextHrzJustification`. The values defined by the enumerated type are as follows:

- `PRO_TEXT_HRZJUST_DEFAULT`—Aligns the text using the default justification. The justification selected for the first note becomes the default for all successive notes added during the current session.
- `PRO_TEXT_HRZJUST_LEFT`—Aligns the text style object to the left.

- PRO_TEXT_HRZJUST_CENTER—Aligns the text style object in the centre.
- PRO_TEXT_HRZJUST_RIGHT—Aligns the text style object to the right

The function ProTextStyleVertJustificationGet() returns the vertical justification for the text style object.

The function ProTextStyleVertJustificationSet() sets the vertical justification for the text style object using the enumerated data type ProVerticalJustification. The values defined by the enumerated type are as follows:

- PRO_VERTJUST_DEFAULT—Aligns the text using the default justification. The justification selected for the first note becomes the default for all successive notes added during the current session.
- PRO_VERTJUST_TOP—Aligns the text style object to the top.
- PRO_VERTJUST_MIDDLE—Aligns the text style object to the middle.
- PRO_VERTJUST_BOTTOM—Aligns the text style object to the bottom.

The function ProTextStyleColorGet() returns the color for the text style object. If the text style object is of default color, the function returns PRO_COLOR_METHOD_TYPE with undefined color. From Creo Parametric 2.0 M200 onward, the function ProTextStyleColorGetWithDef() supersedes the function ProTextStyleColorGet(). The function ProTextStyleColorGetWithDef() also supports default color. If the specified text style object is of default color, PRO_COLOR_METHOD_DEFAULT will be returned.

The function ProTextStyleColorSet() sets the color for the text style object. From Creo Parametric 2.0 M200 onward, the function ProTextStyleColorSetWithDef() supersedes the function ProTextStyleColorSet(). The function ProTextStyleColorSetWithDef() also supports default color and enables you to set the text style object to the default color.

The functions ProTextStyleIsHeightInModelUnits() and ProTextStyleHeightInModelUnitsSet() obtain and assign whether the text height is in relation to the model units, or a fraction of the screen size. These functions are applicable only for flat-to-screen annotations.

The function ProSolidDefaulttextheightGet() returns the default text height for annotations and dimensions for a given solid model.

Accessing the Note Placement

The functions described in this section provide access to the properties of a 3D note.

Functions Introduced:

- **ProNotePlacementGet()**
- **ProNoteAttachFreeGet()**
- **ProSolidDispoutlineGet()**
- **ProNoteAttachLeadersGet()**
- **ProNoteAttachOnitemGet()**
- **ProNoteAttachPlaneGet()**
- **ProNoteAttachRelease()**

The function `ProNotePlacementGet()` retrieves the a `ProNoteAttach` structure for the given note. The `ProNoteAttach` object is an opaque handle that describes the location of a note and the leaders attached to it. The functions in this section enable you to set up a `ProNoteAttach` object and assign it to a note, and to read the `ProNoteAttach` information on a note.

The function `ProNoteAttachFreeGet()` retrieves the location of the note text. The note text is stored in relative model coordinates, where {0.5, 0.5, 0.5} indicates the exact center of the model's display bounding box obtained from `ProSolidDispoutlineGet()`, and {0.0, 0.0, 0.0} and {1.0, 1.0, 1.0} represent the corners of the box.

The function `ProSolidDispoutlineGet()` provides you with the maximum and minimum values of X, Y, and Z occupied by the display outline of the solid, with respect to the default provided coordinate system.

The function `ProNoteAttachLeadersGet()` returns the attachment points and properties for the leaders stored in the specified note attachment data.

Note

The function `ProNoteAttachLeadersGet()` requires the owner of the note to be displayed.

The functions `ProNoteAttachOnitemGet()` provides the location of an "On Item" note attachment data.

The functions `ProNoteAttachPlaneGet()` returns the annotation plane assigned to the note attachment data.

The function `ProNoteAttachRelease()` releases the allocated opaque handle.

Modifying 3D Note Attachments

The actual note created in Creo Parametric will not be modified by the access functions until the note attachment is assigned to the note using the modification function `ProNotePlacementSet()`.

Functions Introduced:

- **ProNoteAttachAlloc()**
- **ProNoteAttachFreeSet()**
- **ProNoteAttachAddend()**
- **ProNoteAttachLeaderAdd()**
- **ProNoteAttachLeaderRemove()**
- **ProNoteAttachOnitemSet()**
- **ProNoteAttachPlaneSet()**
- **ProNotePlacementSet()**

The function `ProNoteAttachAlloc()` allocates a `ProNoteAttach` object for a note attachment.

To set the location of an attachment point, call the function `ProNoteAttachFreeSet()`. See the description of `ProNoteAttachFreeGet()` for an explanation of the coordinates used by this function.

The function `ProNoteAttachAddend()` adds a leader to the specified attachment. The leader points to a location on the parent model specified by an argument of type `ProSelection`.

The attachment types are specified in `ProNoteAttachAttr`. The possible values are as follows:

- `PRO_NOTE_ATT_NONE`
- `PRO_NOTE_ATT_NORMAL`
- `PRO_NOTE_ATT_TANGENT`

Use the function `ProNoteAttachLeaderAdd()` to specify the type of arrowhead for the leader. This function supersedes the function `ProNoteAttachAddend()`.

The function `ProNoteAttachLeaderAdd()` adds a leader to the specified attachment. The leader points to a location on the parent model specified by an argument of type `ProSelection`. The selection UV parameters determine the precise attachment point for the note leader. The attachment types are specified by the parameter *attr* and can have one of the following values:

- `PRO_NOTE_ATT_NORMAL`—Specifies a normal attachment.
- `PRO_NOTE_ATT_TANGENT`—Specifies a tangent attachment.

The function `ProNoteAttachLeaderRemove()` removes a leader from the note attachment data.

The function `ProNoteAttachOnitemSet()` sets the location of an "On Item" note placement. Using this function removes any leaders currently assigned to the note attachment.

The function `ProNoteAttachPlaneSet()` sets the annotation plane of the notes. The annotation plane of a note in an annotation element may not be removed.

The function `ProNotePlacementSet()` assigns the a `ProNoteAttach` structure for the given note, thus defining or redefining the placement for the note.

Note

If modifying an existing note, the functions `ProNoteAttach*Set()` do not modify the note until the note attachment is reassigned to the note using the modification function `ProNotePlacementSet()`.

Geometric Tolerances

For more information on Geometric Tolerances refer to the chapter Annotations: Geometric Tolerances on page 617.

Accessing Set Datum Tags

The functions described in this section provide the ability to access and display set datum tag annotations in 3D models.

Functions Introduced:

- **ProSolidSetdatumtagVisit()**
- **ProSetdatumtagReferenceGet()**
- **ProGeomitemSetdatumtagGet()**
- **ProSetdatumtagAttachmentGet()**
- **ProSetdatumtagAttachmentSet()**
- **ProSetdatumtagPlacementGet()**
- **ProSetdatumtagPlaneGet()**
- **ProSetdatumtagPlaneSet()**
- **ProSetdatumtagTextstyleGet()**
- **ProSetdatumtagTextstyleSet()**

- **ProMdlSetdatumtagCreate()**
- **ProMdlSetdatumtagDelete()**
- **ProSetdatumtagLabelGet()**
- **ProSetdatumtagLabelSet()**
- **ProSetdatumtagAdditionalTextGet()**
- **ProSetdatumtagAdditionalTextSet()**
- **ProSetdatumtagElbowGet()**
- **ProSetdatumtagElbowSet()**
- **ProSetdatumtagASMEDisplayGet()**
- **ProSetdatumtagASMEDisplaySet()**
- **ProSetdatumtagReferencesAdd()**
- **ProSetdatumtagReferencesGet()**
- **ProSetdatumtagReferenceDelete()**
- **ProSetdatumtagTextPointGet()**
- **ProSetdatumtagAdditionalTextLocationGet()**
- **ProDrawingSetDatumTagIsShown()**
- **ProDrawingSetdatumtagErase()**
- **ProDrawingSetdatumtagVisit()**

The function `ProSolidSetdatumtagVisit()` enables you to visit the set datum tag annotations in a solid model.

The function `ProSetdatumtagCreate()` is deprecated. Use the function `ProMdlSetdatumtagCreate()` instead. The function `ProSetdatumtagCreate()` creates a new set datum tag annotation. The input arguments are as follows:

- *reference*—Specify a datum reference (plane or axis) from the model as the geometric reference for the set datum tag. If you want to make this tag reference model geometry, use Creo Parametric TOOLKIT to create a datum plane or axis referencing the geometry first, and then use that datum in this argument.
- *annotation_plane*—Specify an Annotation Plane for the annotation.
- *attachment*—Optionally, specify the location for placement of the set datum tag. The argument can contain:
  - A dimension
  - A gtol.
  - A geometry selection to which the set datum will be attached.

Note

Once the set datum tag annotation is created, use the function `ProAnnotationShow()` to display it.

The function `ProSetdatumtagAttachmentGet()` returns the item on which the datum tag is placed.

The function `ProSetdatumtagAttachmentSet()` specifies the item on which the datum tag is placed.

Note

From Creo Parametric 2.0 M090 onward, to specify the datum plane or datum axis as the attachment option, pass the input argument `attachment` as *NULL* in the function `ProSetdatumtagAttachmentSet()`. The datum tag is attached to the datum axis at the default location.

The function `ProSetdatumtagPlacementGet()` returns the item type, id, and owner on which the set datum tag is placed. Use this function in cases where it is not possible to construct the selection. For example, when the solid owned datum feature symbol is attached to a solid owned dimension that is created in a drawing. The function returns the attachment item contained in the `ProModelitem` structure.

The function `ProSetdatumtagPlaneGet()` returns the annotation plane for the specified set datum tag.

The function `ProSetdatumtagPlaneSet()` sets the annotation plane for the set datum tag.

The function `ProGeomitemSetdatumtagGet()` returns the set datum tag that uses the specified geometric item as the reference, if available.

The function `ProSetdatumtagReferenceGet()` returns the datum specified as a reference for the set datum tag.

Note

The functions `ProGeomitemSetdatumtagGet()` and `ProSetdatumtagReferenceGet()` return information only for datum tags created in releases prior to Creo Parametric 4.0 F000.

The function ProSetdatumtagTextstyleGet() returns the text style details for the set datum tag. The function ProSetdatumtagTextstyleGet() is deprecated. Use the function ProAnnotationTextstyleGet() instead.

The function ProSetdatumtagTextstyleSet() sets the text style details for the set datum tag. The function ProSetdatumtagTextstyleSet() is deprecated. Use the function ProAnnotationTextstyleSet() instead.

The function ProMdlSetdatumtagCreate() creates a datum feature symbol in the specified drawing or solid. After creating a datum feature symbol, to display it, call the function ProAnnotationShow(). The input arguments are as follows:

- *p_mdl*—Specifies a drawing or solid.
- *attachment*—Specifies the location of the datum feature symbol on the geometry. The argument can contain:
  - A dimension
  - A gtol
  - A geometry selection to which the set datum will be attached
- *annotation_plane*—Specifies an annotation plane for the annotation. If the datum feature symbol is attached to a dimension or gtol, the argument can be NULL. However, for a drawing, the argument must be passed as NULL.
- *label*—Specifies the label for the datum feature symbol.

Use the function ProMdlSetdatumtagDelete() to delete the specified datum feature.

You can specify a string identifier as a label that will appear within the datum feature symbol frame. The functions ProSetdatumtagLabelGet() and ProSetdatumtagLabelSet() get and set the label for the specified datum feature symbol.

You can also specify additional text that appears along with the datum feature symbol instance. The label for the datum feature symbol and the additional text appear horizontally aligned to the screen. The functions ProSetdatumtagAdditionalTextGet() and ProSetdatumtagAdditionalTextSet() get and set the additional text for the specified datum feature symbol. The functions also enable you to get and set the position of the additional text around the frame of the datum feature symbol using the enumerated data type ProDtmFeatAddlTextPos. The valid values are:

- PRO_DTM_FEAT_ADDL_TEXT_RIGHT
- PRO_DTM_FEAT_ADDL_TEXT_BOTTOM
- PRO_DTM_FEAT_ADDL_TEXT_LEFT

- PRO_DTM_FEAT_ADDL_TEXT_TOP
- PRO_DTM_FEAT_ADDL_TEXT_DEFAULT

It is possible to set the display style of the leader which attaches the datum feature symbol to the geometry. You can set the appearance of the leader as straight or have an elbow. The function ProSetdatumtagElbowSet() enables you set the display of the leader. Pass the input argument *elbow* as PRO_B_TRUE to set the leader with an elbow. The function ProSetdatumtagElbowGet() checks if the leader of the specified datum feature symbol has an elbow.

The function ProSetdatumtagASMEDisplaySet displays the datum feature symbol according to the ASME standard. Use the function ProSetdatumtagASMEDisplayGet() to check if the specified datum feature symbol is displayed as per ASME standard.

The function ProSetdatumtagReferencesAdd() adds semantic references to the specified datum feature symbol.

Note

When a reference includes more than one collection, the function ProSetdatumtagReferencesAdd() returns the error PRO_TK_MAX_LIMIT_REACHED and no reference is added.

The function ProSetdatumtagReferencesGet() returns a ProArray of semantic references that were used to place the specified datum feature symbol. Use the function ProAnnotationreferencearrayFree to free the allocated memory. Use the function ProSetdatumtagReferenceDelete() to delete semantic references in the specified datum feature symbol. The references are specified by their index number.

In Creo Parametric 7.0.0.0 and later, the function ProSetdatumtagTextLocationGet() is deprecated.

Use the function ProSetdatumtagTextPointGet() instead. The function ProSetdatumtagTextPointGet() retrieves the text point for the specified datum feature symbol.

The function ProSetdatumtagAdditionalTextLocationGet() retrieves the location of additional text for the specified datum feature symbol.

The function ProDrawingSetDatumTagIsShown() returns the display status of the set datum tag in the specified view of a drawing. The input arguments are:

- *set_datum_tag*—Specifies the set datum tag.
- *drawing*—Specifies the drawing that shows the annotation.
- *view*—Specifies the drawing view.

The function returns PRO_B_TRUE if the set datum tag is shown in the specified drawing view, and PRO_B_FALSE if it is not shown in the drawing view.

The function ProDrawingSetDatumTagIsShown() returns the error PRO_TK_BAD_CONTEXT if the datum feature symbol cannot be shown in the specified drawing view.

Use the function ProDrawingSetdatumtagErase() to set a set datum tag to be erased from the specified view of a drawing. The annotation is not displayed until it is explicitly displayed using the function ProAnnotationShow().

The function ProDrawingSetdatumtagVisit() enables you to visit the set datum tag annotations in the specified drawing.

Accessing Set Datums for Datum Axes or Planes

The functions described in this section provide access to the “Set Datum” status of a datum axis or plane.

Note

These function support the “Set Datum” capability which existed before Set Datum Tag annotations.

Functions Introduced:

- **ProGeomitemSetdatumGet()**
- **ProGeomitemSetdatumSet()**
- **ProGeomitemSetdatumClear()**

The function ProGeomitemSetdatumGet() specifies whether the datum plane or axis is a “Set Datum”. This function supersedes the function ProGeomitemIsGtolref().

The function ProGeomitemSetdatumSet() sets the datum plane or axis to be a “Set Datum”. This function supersedes the function ProGeomitemGtolrefSet().

The function ProGeomitemSetdatumClear() removes the “Set Datum” status of a datum plane or axis. This function supersedes the function ProGeomitemGtolrefClear().

Surface Finish Annotations

The functions described in this section provide read access to the properties of the surface finish object. They also allow you to create and modify surface finishes.

The style of surface finishes for releases previous to Pro/ENGINEER Wildfire 2.0 was a flat-to-screen symbol attached to a single surface. From Pro/ENGINEER Wildfire 2.0 onwards, the method for construction of surface finishes has been modified. The new style of surface finish is a symbol instance that may be attached on a surface or with a leader. The following functions support both the old and new surface finish annotations, except where specified.

Functions Introduced:

- **ProSolidSurffinishVisit()**
- **ProSurffinishCreate()**
- **ProSurffinishReferencesGet()**
- **ProSurffinishSrfcollectionGet()**
- **ProSurffinishSrfcollectionSet()**
- **ProSurffinishNameGet()**
- **ProSurffinishNameSet()**
- **ProSurffinishDataGet()**
- **ProSurffinishModify()**
- **ProSurffinishValueGet()**
- **ProSurffinishValueSet()**
- **ProSurffinishDelete()**

The function `ProSolidSurffinishVisit()` visits the surface finishes stored in the specified solid model. This function accepts a visit function `ProSurfaceVisitAction()` and a filter function `ProSurfaceFilterAction()` as the input arguments.

The function `ProSurffinishCreate()` creates a new symbol-based surface finish annotation. The function requires a symbol instance data structure for creation. Once the surface finish annotation is created, use the function `ProAnnotationShow()` to display it.

Note

The data must conform to the requirements for surface finishes, that is, attachment must be via leader to or on one or more surfaces.

You can use the standard surface finish symbol definitions from the symbol instance data structure. Use the function `ProSolidDtlsymdefRetrieve()` to retrieve the surface finish symbol definitions from the location `PRO_DTLSYMDEF_SRC_SURF_FINISH_DIR`. The surface finish value should be set using the variant text options for symbol instances.

The function `ProSurffinishReferencesGet()` returns the surface or surfaces referenced by the surface finish.

The function `ProSurffinishSrfcollectionGet()` obtains a surface collection which contains the references of the surface finish.

The function `ProSurffinishSrfcollectionSet()` assigns a surface collection to be the references of the surface finish. This overwrites all current surface finish references. The following types of surface collections are supported:

- One by one surface set
- Intent surface set
- Excluded surface set
- Seed and Boundary surface set
- Loop surface set
- Solid surface set
- Quilt surface set

Note

Only those surface finishes that are contained within annotation elements may use a collection of references instead of a single surface reference.

The function `ProSurffinishNameGet()` returns the name of the surface finish annotation.

The function `ProSurffinishNameSet()` sets the name of the surface finish annotation.

The function `ProSurffinishDataGet()` returns the symbol instance data for the surface finish. This function supports only new symbol-based surface finishes.

The function `ProSurffinishModify()` modifies the symbol instance data for the specified surface finish. This function supports only new symbol-based surface finishes.

The function `ProSurffinishValueGet()` retrieves the value of a surface finish annotation.

The function `ProSurffinishValueSet()` sets the value of a surface finish annotation.

The function `ProSurffinishDelete()` deletes the specified surface finish.

The functions described above supersede the Pro/Develop functions `pro_surf_finish_number()`, `pro_get_surf_finish()`, `pro_delete_surface_finish()` and `pro_set_surface_finish()`.

Symbol Annotations

The functions described in this section provide support for 3D mode symbols. Creo Parametric TOOLKIT functions for symbol instances are used in both 2D and 3D modes. Symbols for a particular mode must conform to the requirements for that mode. Some `ProDtlsyminst` functions have been modified to accept a `ProMdl` object as the input instead of `ProDrawing` object to support 3D mode operations.

Creating, Reading and Modifying 3D Symbols

Functions Introduced:

- **ProDtlsyminstCreate()**
- **ProDtlsyminstDataGet()**
- **ProDtlsyminstModify()**
- **ProDtlsyminstdataPlaneGet()**
- **ProDtlsyminstdataPlaneSet()**

The function `ProDtlsyminstCreate()` creates a symbol instance in the specified model. Once the symbol instance is created, use the function `ProAnnotationShow()` to display it.

The function `ProDtlsyminstDataGet()` returns the symbol instance data.

The function `ProDtlsyminstModify()` modifies the symbol instance.

For more information about creating, accessing and modifying symbols via the `ProDtlsyminst` and `ProDtlsyminstdata` structures, refer to the Drawings on page 1226 chapter.

The functions `ProDtlsyminstdataPlaneGet()` and `ProDtlsyminstdataPlaneSet()` provide access to the symbol annotation plane. Annotation planes are required for 3D symbol instances but are not applicable for 2D symbol instances.

Locating and Collecting 3D Symbols and Symbol Definitions

Functions Introduced:

- **ProSolidDtlsymdefVisit()**
- **ProSolidDtlsyminstVisit()**
- **ProSolidDtlsymdefEntityVisit()**
- **ProSolidDtlsymdefNoteVisit()**
- **ProSolidDtlsymdefRetrieve()**
- **ProSolidDtlsyminstsCollect()**
- **ProSolidDtlsymdefsCollect()**

The functions `ProSolidDtlsymdefVisit()` and `ProSolidDtlsyminstVisit()` allow traversal of symbol definitions and instances in a solid model.

The functions `ProSolidDtlsymdefEntityVisit()` and `ProSolidDtlsymdefNoteVisit()` allow traversal of items contained in a symbol definition stored in a solid model.

The function `ProSolidDtlsymdefRetrieve()` allows retrieval of a symbol definition into a given solid model. The input arguments for this function are as follows:

- *solid*—Specifies a handle to the solid model.
- *location*—Specifies the location of the symbol definition file. It can one of the following values:
  - `PRO_DTLSYMDEF_SRC_SYSTEM`—Specifies the system symbol definition directory.
  - `PRO_DTLSTMDEF_SRC_SYMBOL_DIR`—Specifies the system surface finish symbol definition directory.
  - `PRO_DTLSYMDEF_SRC_SYMBOL_DIR`—Specifies the location controlled by the configuration option `pro_symbol_dir`.
  - `PRO_DTLSYMDEF_SRC_PATH`—Specifies the absolute path to a directory containing the symbol definition.
- *filepath*—Specifies the path to the file with a symbol definition. The path is relative to the location specified in the argument *location*.
- *filename*—Specifies the name of the symbol definition file.

- *version*—Specifies the version of the symbol definition file.
- *update*—Specifies the update flag. If `TRUE`, the definition will be loaded even if a definition of that name already exists in the model. If `FALSE`, the retrieval will not take place if the definition exists in the model.

You can use symbol instances from different symbol definitions in a solid. The function `ProSolidDtlsyminstsCollect()` collects all the symbol instances used in the specified solid as a `ProArray`. The function `ProSolidDtlsymdefsCollect()` returns a `ProArray` of all the symbol definitions used in the specified solid. The function returns symbol definitions only for the symbol instances used in the solid. Use the function `ProArrayFree()` to release the memory assigned to the `ProArray` of symbol definitions and symbol instances.

25

Annotations: Geometric Tolerances

The functions in this chapter allow a Creo Parametric TOOLKIT application to read, modify, and create geometric tolerances (gtols) in a solid or drawing. We recommend that you study the Creo Parametric documentation on geometric tolerances, and develop experience with manipulating geometric tolerances using the Creo Parametric commands before attempting to use these functions.

Geometric Tolerance Objects

Overview

The geometric tolerance objects enable you to access internal data structure of geometric tolerances (gtol). The object also references and gets attachment details for gtols.

From Creo Parametric 4.0 F000 onward, the object `Gtoldata` has been deprecated. All the `ProGtolData*` functions have also been deprecated. Use the new `ProGtol*` functions instead. The new functions are defined in the header files `ProGtol.h` and `ProGtolAttach.h`.

Note

Geometric tolerance functions deprecated in Creo Parametric 4.0 F000 must not be used with the new geometric tolerance functions available from Creo Parametric 4.0 F000 in a Creo Parametric TOOLKIT application. If the functions are used together in an application, the results may be unpredictable.

ProGtol

Geometric tolerances in a Creo Parametric model are referenced by the data handle, `ProGtol`. This handle is identical to `ProModelitem`, in which the `type` field is set to `PRO_GTOL`. You can use `ProSelect()` with the option `gtol` to select a gtol, after which you can extract the `ProGtol` handle using `ProSelectionModelitemGet()`.

ProGtolAttach

This is an opaque handle object that references an internal data structure which provides complete attachment details for a gtol. The structure contains attachment information such as, type of placement, annotation plane, location and references and so on for a gtol.

Visiting Geometric Tolerances

Function Introduced:

- **ProMdlGtolVisit()**

The function `ProMdlGtolVisit()` visits geometric tolerances stored in a part, assembly, or drawing. The forms of the visit and filter functions are similar to those of most other visit functions-they receive a `ProGtol` pointer as input argument to identify the gtol.

Reading Geometric Tolerances

The functions explained in this section enable you to access and read the properties of a geometric tolerance.

Function Introduced:

- **ProGtolNameGet()**
- **ProGtolTypeGet()**
- **ProGtolTopModelGet()**
- **ProGtolReferencesGet()**
- **ProGtolDatumReferencesGet()**
- **ProGtolValueStringGet()**
- **ProGtolCompositeGet()**
- **ProGtolCompositeShareRefGet()**
- **ProGtolSymbolStringGet()**
- **ProGtolIndicatorsGet()**
- **ProGtolAllAroundGet()**
- **ProGtolAllOverGet()**
- **ProGtolAddlTextBoxedGet()**
- **ProGtolBoundaryDisplayGet()**
- **ProGtolUnilateralGet()**

The function `ProGtolNameGet()` returns the name of the geometric tolerance (gtol) as a `wchar_t*` string. Use the function `ProWstringFree()` to free the string.

The function `ProGtolTypeGet()` returns the type of the gtol using the enumerated data type `ProGtolType`. The various types of gtol, are straightness, flatness, and so on.

The function `ProGtolTopModelGet()` returns the model that defines the origin of `ProSelection` structures used to define references inside the gtol. This will usually be the model that contains the gtol; but if the gtol was created in drawing mode and added to a solid in a drawing view, the owner will be the drawing, while the model is the solid.

ProGtolReferencesGet() returns a ProArray of the geometric entities referenced by the specified gtol. The entities are additional references used to create the gtol.

Use the function ProAnnotationreferencearrayFree() to free the ProArray.

The function ProGtolDatumReferencesGet() returns the primary, secondary, and tertiary datum references for a gtol as wchar_t* strings. Use the function ProWstringFree() to free the strings.

The function ProGtolCompositeGet() retrieves the value and the datum references, that is, the primary, secondary, and tertiary references for the specified composite gtol.

The function ProGtolCompositeGet() returns an array of values in which the first value is the primary value of the gtol and the rest are secondary, tertiary, and so on.

The function ProGtolValueStringGet() retrieves the value specified in the gtol as a wchar_t* string. Use the function ProWstringFree() to free the string.

The function ProGtolValueStringGet() returns a value string which is the primary value of the gtol and is displayed in the first box of the **Composite Frame** in the gtol ribbon.

For more information about gtols and the ribbon tab, refer to the Creo Parametric online help.

Use the function ProGtolSymbolStringGet() to retrieve the string value in the specified font for the gtol symbol. The input arguments are:

- *symbol*—Specifies the type of symbol using the enumerated data type ProGtolSymbol. The valid values are:
  - PRO_GTOL_SYMBOL_DIAMETER—Specifies the diameter for the feature.
  - PRO_GTOL_SYMBOL_FREE_STATE—Specifies that the model is not subjected to any force, except the gravitational force.
  - PRO_GTOL_SYMBOL_STAT_TOL—Specifies statistical tolerance.
  - PRO_GTOL_SYMBOL_TANGENT_PLANE—Specifies the tangent plane.
  - Material Symbols—The material conditions are represented using the following values:

    PRO_GTOL_SYMBOL_LMC—Specifies least material condition and is displayed by the symbol 

    .

- PRO_GTOL_SYMBOL_MMC—Specifies maximum material condition and is displayed by the symbol  .
- PRO_GTOL_SYMBOL_RFS—Specifies the material condition regardless of the feature size and is displayed by the symbol  .
- PRO_GTOL_SYMBOL_DEFAULT_RFS—Specifies the material condition regardless of the feature size, but does not show a symbol in the frame.
- PRO_GTOL_SYMBOL_LMC_R—Specifies least material condition with reciprocity. The reciprocity is displayed on drawings with the symbol ® after the symbol  .
- PRO_GTOL_SYMBOL_MMC_R—Specifies maximum material condition with reciprocity. The reciprocity is displayed on drawings with the symbol ® after the symbol  .

Note

You can use the reciprocity condition together with the maximum material condition or the minimum material condition to use the maximum tolerance allowed for a feature. The material conditions with reciprocity are applicable only for the ISO standard.

- Indicator Symbols—Indicators are displayed after the gtol symbol according to the standard. The Indicator symbols are represented using the following values:

  PRO_INDICATOR_SYMBOL_ANGULARITY—Specifies the angular position of a feature to a reference.
- PRO_INDICATOR_SYMBOL_PERPENDICULARITY—Specifies that the two features must be perpendicular to each other. It is also used to indicate perpendicularity for features such as axis to a reference.
- PRO_INDICATOR_SYMBOL_PARALLELISM—Specifies that the two features must be parallel to each other. It is also used to indicate parallelism for features such as axis to a reference.
- PRO_INDICATOR_SYMBOL_SYMMETRY—Specifies that the two features must be symmetric about a center.
- PRO_INDICATOR_SYMBOL_RUNOUT—Applicable only for direction

feature. Specifies that the direction of the width of the tolerance zone is equal to run-out, that is perpendicular to the surface of the tolerance feature.

- *font*—Specifies the tolerancing font used for symbols. The valid values are defined in the enumerated data `ProSymbolFont`:
  - `PRO_FONT_LEGACY`—Specifies that symbols use the legacy font.
  - `PRO_FONT_ASME`—Specifies that symbols use ASME font.
  - `PRO_FONT_ISO`—Specifies that symbols use ISO font.
  - `PRO_FONT_STROKED_ASME`—Specifies that symbols use stroked ASME fonts that are native to Creo.
  - `PRO_FONT_STROKED_ISO`—Specifies that symbols use stroked ISO fonts that are native to Creo.

The function `ProGtolIndicatorsGet()` retrieves all the indicators assigned to the specified gtol. It returns `ProArray` of indicator types, symbols, and datum feature symbols.

The function `ProGtolCompositeShareRefGet()` checks if the datum references are shared between all the rows defined in the composite gtol.

The functions `ProGtolSymbolStringGet()` and `ProGtolValueStringGet()`, provide information on the different symbolic modifiers available to the gtol.

The function `ProGtolAllOverGet()` returns a boolean value that indicates if the **All Over** symbol has been set in the specified gtol. The function `ProGtolAllAroundGet()` checks if the **All Around** symbol has been set for the specified gtol. The **All Over** symbol and **All Around** symbol specifies that the profile tolerance must be applied to all the three dimensional profile of the part. The symbol is available only for surface profile gtol, that is, of type `PROGTOLTYPE_SURFACE`.

The function `ProGtolAddlTextBoxedGet()` checks if a box has been created around the specified additional text in a geometric tolerance.

The function `ProGtolBoundaryDisplayGet()` checks if the boundary modifier has been set for the specified gtol. Use the function `ProGtolUnilateralGet()` to check if the profile boundary has been set to unilateral in the specified gtol. If set to unilateral, the function also checks if the tolerance disposition is in the outward direction of the profile.

Note

When the new `ProGtol*Get()` functions, except `ProGtolTopModelGet()`, `ProGtolReferencesGet()`, `ProGtolReferenceDelete()`, and `ProGtolValidate()` are called on geometric tolerances created in releases prior to Creo Parametric 4.0 F000, these legacy geometric tolerances are converted to the new Creo Parametric 4.0 geometric tolerances. In this case, the revision number of the model is also incremented.

Creating a Geometric Tolerance

Functions Introduced:

- **ProGtolTypeSet()**
- **ProGtolReferencesAdd()**
- **ProGtolReferenceDelete()**
- **ProGtolDatumReferencesSet()**
- **ProGtolValueStringSet()**
- **ProGtolCompositeSet()**
- **ProGtolCompositeShareRefSet()**
- **ProGtolAllAroundSet()**
- **ProGtolAllOverSet()**
- **ProGtolAddlTextBoxedSet()**
- **ProGtolElbowlengthSet()**
- **ProGtolBoundaryDisplaySet()**
- **ProGtolUnilateralSet()**
- **ProMdlGtolCreate()**
- **ProGtolDtlnotesCollect()**

The basic steps in creating a gtol are:

1. Allocate a `ProGtolAttach` structure using `ProGtolAttachAlloc()`.
2. Set the attachment properties using the ProGtolAttach*Set() functions.
3. Set tolerance properties using the `ProGtol*Set()` functions.

4. Create the tolerance using `ProMdlGtolCreate()`. Once the tolerance is created, you must use the function `ProAnnotationShow()` to display it.
5. Free `ProGtolAttach` using `ProGtolAttachFree()`.

The function `ProGtolTypeSet()` sets the type of geometric tolerance using the enumerated data type `ProGtolType`.

Use the function `ProGtolReferencesAdd()` to add datum references to the specified gtol.

Note

When a reference includes more than one collection, the function `ProGtolReferencesAdd()` returns the error `PRO_TK_MAX_LIMIT_REACHED` and no reference is added.

Use the function `ProGtolReferenceDelete()` to delete the datum references from the specified gtol.

The function `ProGtolIndicatorsSet()` sets the indicators for the specified gtol. The input arguments are:

- *gtol*—Specifies the gtol.
- *types*—Specifies a `ProArray` of indicator types using the enumerated data type `ProGtolIndicatorType`. Use the function `ProArrayFree()` to free the array.
- *symbols*—Specifies a `ProArray` of strings for indicator symbols. Free the array using the function `ProWstringproarrayFree()`.
- *dfs*—Specifies a `ProArray` of strings for datum feature symbols. Free the array using the function `ProWstringproarrayFree()`.

The function `ProGtolCompositeSet()` sets the value and datum references, primary, secondary, and tertiary for the specified composite gtol.

Use the function `ProGtolCompositeShareRefSet()` to specify if datum references in a composite gtol must be shared between all the defined rows. Pass the input argument *share* as `PRO_B_TRUE` to share the references.

The function `ProGtolValueStringSet()` sets the specified value for a gtol.

Use the function `ProGtolDatumReferencesSet()` to set the datum references for the specified gtol. The datum references are set as `wchar_t*` strings. Use the function `ProWstringFree()` to free the strings.

The datum references are given to `ProGtolDatumReferencesSet()` in the form of `wchar_t*` strings. Use the function `ProWstringFree()` to free the strings.

The function `ProGtolElbowlengthSet()` sets the elbow along with its properties for a leader type of gtol. The function is supported for leader type gtols which are placed on the annotation plane. The input arguments are:

- *gtol*—Specifies a gtol.
- *elbow_length*—Specifies the length of the elbow in model coordinates.
- *elbow_direction*—Specifies the direction of the elbow in model coordinates. The gtol text also moves in this direction.

The function `ProGtolBoundaryDisplaySet()` sets the boundary modifier for the specified gtol. Use the function `ProGtolUnilateralSet()` to set the profile boundary as unilateral in the specified gtol. The function also sets the tolerance disposition to the outward direction of the profile.

The function `ProGtolAllAroundSet()` sets the **All Over** symbol for the specified geometric tolerance.

The function `ProGtolAllAroundSet()` sets the **All Around** symbol for the specified geometric tolerance.

The function `ProGtolAddlTextBoxedSet()` creates a box around the specified additional text in a geometric tolerance. Boxes can be created around additional text added above and below the frame of the geometric tolerance.

From Creo Parametric 4.0 F000 onward, stacked geometric tolerance creates separate notes for each tolerance. For a stacked geometric tolerance, the functions `ProGtolDtlnoteGet()`, `ProDtlnoteDataGet()`, and `ProDtlnotedataLinesCollect()` will return information about individual notes for each geometric tolerance.

The function `ProGtolDtlnotesCollect()` returns the detail notes that represent a geometric tolerance in the specified drawing. The input arguments follow:

- *solid_model_gtol*—The handle to the geometric tolerance.
- *drawing* —The drawing where the note is displayed.

The function returns an array of drawing notes that represents the geometric tolerance.

Attaching the Geometric Tolerances

The functions explained in this section enable you to access and set attachment options for the geometric tolerance.

Function Introduced:

- **ProGtolAttachGet()**
- **ProGtolAttachSuppressedLeadersGet()**
- **ProGtolAttachSet()**

- **ProGtolAttachTypeGet()**
- **ProGtolAttachLeadersGet()**
- **ProGtolAttachLeadersSet()**
- **ProGtolleaderGet()**
- **ProGtolleaderZExtensionlineGet()**
- **ProGtolEnvelopeGet()**
- **ProGtolleadersFree()**
- **ProGtolAttachOffsetItemGet()**
- **ProGtolAttachOffsetItemSet()**
- **ProGtolAttachFreeGet()**
- **ProGtolAttachFreeSet()**
- **ProGtolleaderFree()**
- **ProGtolAttachOnDatumGet()**
- **ProGtolAttachOnDatumSet()**
- **ProGtolAttachOnAnnotationGet()**
- **ProGtolAttachOnAnnotationSet()**
- **ProGtolAttachMakeDimGet()**
- **ProGtolAttachMakeDimSet()**

The function `ProGtolAttachGet()` retrieves all the attachment related information for a gtol as a `ProGtolAttach` structure. Use the function `ProGtolAttachSet()` to set the attachment options. If the function `ProGtolAttachSet()` specifies one or more leaders, the leaders are described by a separate opaque object called `ProGtolleader`. This object is allocated by call to the function `ProGtolleaderAlloc()`. Use the function `ProGtolleaderFree()` to free the allocated memory.

The function `ProGtolAttachTypeGet()` retrieves the type of attachment for a gtol. It uses the enumerated data type `ProGtolAttachType` to provide information about the placement of the gtol. The valid values are:

- `PRO_GTOL_ATTACH_DATUM` – Specifies that the gtol is placed on its reference datum.
- `PRO_GTOL_ATTACH_ANNOTATION`—Specifies that the gtol is attached to an annotation.
- `PRO_GTOL_ATTACH_ANNOTATION_ELBOW`—Specifies that the gtol is attached to the elbow of an annotation.
- `PRO_GTOL_ATTACH_FREE`—Specifies that the gtol is placed as a free. It is unattached to the model or drawing.

- PRO_GTOL_ATTACH_LEADERS—Specifies that the gtol is attached with one or more leader to geometry such as, edge, dimension witness line, coordinate system, axis center, axis lines, curves, or surface points, vertices, section entities, draft entities, and so on. The leaders are represented using an opaque handle, ProGtolleader.
- PRO_GTOL_ATTACH_OFFSET—Specifies that the gtol frame can be placed at an offset from the following drawing objects: dimension, dimension arrow, gtol, note, and symbol.
- PRO_GTOL_ATTACH_MAKE_DIM—Specifies that the gtol frame is attached to a dimension line.

Use the function ProGtolAttachLeadersGet() to get attachment details for leader type of gtol. The output arguments are:

- *plane*—Specifies the annotation plane. For gtols defined in drawing, it returns NULL.
- *type*—Specifies the attachment type for the leader using the enumerated data type ProGtolLeaderAttachType.
- *leaders*—Specifies a ProArray of gtol leaders.
- *location*—Specifies the location of gtol text in model coordinates.

Use the function ProGtolAttachSuppressedLeadersGet() to get the number of leaders that are suppressed due to missing references.

Use the function ProGtolAttachLeadersSet() to set the attachment options for leader type of gtol.

To unpack the information in the ProGtolleader handle, use the function ProGtolleaderGet(). After reading the leaders, free the leader array by calling ProGtolleadersFree().

The function ProGtolleaderZExtensionlineGet() retrieves the Z-Extension line of the gtol leader. The leader location coordinates are required when the gtol is moved to a different annotation plane.

The function ProGtolEnvelopeGet() returns the envelope of the gtol. The output argument envelope is the envelope surrounding the gtol in the coordinate system of the model. For drawing, the envelope of the gtol is in the screen coordinates. While retrieving coordinates of the gtol in a specified solid, if the gtol is displayed in the solid as well as in the drawing, the drawing must not be active.

The function ProGtolAttachOffsetItemGet() returns the offset references for the specified ProGtolAttach structure. The function returns the following output arguments:

- *offset_ref*—Specifies the offset reference as a ProSelection object. The reference can be a dimension, arrow of a dimension, another geometric tolerance, note, or a symbol instance. If there are no offset references, the output argument returns NULL.
- *offset*—Specifies the position of the offset reference as model coordinates.

Use the function ProGtolAttachOffsetItemSet() to set the offset references for the specified ProGtolAttach structure.

The function ProGtolAttachFreeGet() gets the details for free type of gtol. It retrieves information about the annotation plane and location of the gtol text in model coordinates. For gtols defined in drawing, the function returns NULL for annotation plane. Use the function ProGtolAttachFreeSet() to set the options for free type gtol.

The functions ProGtolAttachOnDatumGet() and ProGtolAttachOnDatumSet() get and set datum symbol for the geometric tolerance. From Creo Parametric 4.0 F000 onward, datum symbols are defined using datum feature symbol. The functions work with the new datum feature symbol along with the legacy datum tag annotations.

The function ProGtolAttachOnAnnotationGet() retrieves the annotation for the specified ProGtolAttach structure. The function ProGtolAttachOnAnnotationSet() sets the specified annotation to the attachment structure. The input arguments are:

- *gtol_attach*—Specifies the attachment structure ProGtolAttach for a geometric tolerance.
- *p_annot*—Specifies the annotation. For gtols in the solid you can set PRO_DIMENSION, PRO_GTOL, and PRO_NOTE type of annotations. For drawing gtols, you can set PRO_DIMENSION, PRO_GTOL, and PRO_NOTE type of annotations.
- *elbow*—Specifies that the annotation must be placed on the elbow of the leader instead of the gtol text. If the annotation type is set as PRO_NOTE, then you must set *elbow* as PRO_B_TRUE.

The function ProGtolAttachMakeDimGet() gets all the information for a geometric tolerance created with **Make Dim** type of reference. **Make Dim** type of reference mode enables you to create a dimension line and place the gtol frame

attached to it. The geometric tolerance appears in standard dimension format, but with the geometric tolerance instead of a dimension value. The output arguments are:

- *plane*—Specifies the annotation plane for the gtol.
- *attachments_arr*—Specifies the points on the model or drawing where the gtol is attached.
- *dsense_arr*—Specifies more information about how the gtol attaches to each attachment point of the model or drawing.
- *orient_hint*—Specifies the orientation of the gtol using the enumerated data type `ProDimOrient`. The valid values are
  - `PRO_DIM_ORNT_HORIZ`—Specifies a horizontal dimension.
  - `PRO_DIM_ORNT_VERT`—Specifies a vertical dimension.
  - `PRO_DIM_ORNT_SLANTED`—Specifies the shortest distance between two attachment points. This value is available only when the dimension is attached to points.
  - `PRO_DIM_ORNT_ELPS_RAD1`—Specifies the start radius for a dimension on an ellipse.
  - `PRO_DIM_ORNT_ELPS_RAD2`—Specifies the end radius for a dimension on an ellipse.
  - `PRO_DIM_ORNT_ARC_ANG`—Specifies the angle of the arc for a dimension of an arc.
  - `PRO_DIM_ORNT_ARC_LENGTH`—Specifies the length of the arc for a dimension of an arc.
  - `PRO_DIM_ORNT_LIN_TANCRV_ANG`—If the dimension is attached to a line and an end point of a curve, the default dimension will be a linear dimension showing the distance between the line and the curve point. Set this value if you want the dimension to show instead the angle between the line and the tangent at the curve point.
  - `PRO_DIM_ORNT_RAD_DIFF`—Specifies the linear dimension of the radial distance between two concentric arcs or circles.
  - `PRO_DIM_ORNT_NORMAL`—Specifies the linear dimension between two points to be placed normal to the selected reference.
  - `PRO_DIM_ORNT_PARALLEL`—Specifies the linear dimension between two points to be placed parallel to the selected reference.
- *location*—Specifies the location of gtol text as model coordinates.

The function `ProGtolAttachMakeDimSet()` sets all the options to create a geometric tolerance with **Make Dim** type of reference.

Deleting a Geometric Tolerance

Function Introduced:

- **ProGtolDelete()**

The function `ProGtolDelete` permanently removes a gtol.

Validating a Geometric Tolerance

Function Introduced:

- **ProGtolValidate()**

The function `ProGtolValidate` checks if the specified geometric tolerance is syntactically correct. For example, when a string is specified instead of a number for a tolerance value, it is considered as syntactically incorrect. The input arguments are:

- *gtol*—Specifies the geometric tolerance to be checked.
- *ProGtolValidityCheckType*—Specifies the type of check. Currently, the tolerance is checked for correct syntax.

Geometric Tolerance Layout

The functions described in this section provide access to the layout for the text and symbols in a geometric tolerance.

Functions Introduced:

- **ProGtolElbowlengthGet()**
- **ProGtolLineEnvelopeGet()**
- **ProGtolRightTextEnvelopeGet()**

The function `ProGtolElbowlengthGet()` returns the length and direction of the geometric tolerance leader elbow.

The function `ProGtolLineEnvelopeGet()` returns the bounding box coordinates for one line from the geometric tolerance.

The function `ProGtolRightTextEnvelopeGet()` returns the bounding box coordinates for the right text in a specified geometric tolerance.

Note

The functions `ProGtolLineEnvelopeGet()` and `ProGtolRightTextEnvelopeGet()` support the geometric tolerances placed on annotation planes.

Additional Text for Geometric Tolerances

You can place multi-line additional text to the right, left, bottom, and above a geometric tolerance control frame while creating and editing a gtol in both drawing and model modes.

Functions Introduced:

- **ProGtolRightTextGet()**
- **ProGtolRightTextSet()**
- **ProGtolLeftTextGet()**
- **ProGtolLeftTextSet()**
- **ProGtolTopTextGet()**
- **ProGtolTopTextSet()**
- **ProGtolBottomTextGet()**
- **ProGtolBottomTextSet()**

The function `ProGtolRightTextGet()` retrieves the text added to the right of the specified geometric tolerance.

Use the function `ProGtolRightTextSet()` to set the text to be added to the right of the specified geometric tolerance.

The function `ProGtolLeftTextGet()` retrieves the text added to the left of the specified geometric tolerance.

Use the function `ProGtolLeftTextSet()` to set the text to be added to the left of the specified geometric tolerance.

The function `ProGtolTopTextGet()` retrieves the text added to the top of the specified geometric tolerance.

Use the function `ProGtolTopTextSet()` to set the text to be added to the top of the specified geometric tolerance.

The function `ProGtolBottomTextGet()` retrieves the text added to the bottom of the specified geometric tolerance.

Use the function `ProGtolBottomTextSet()` to set the text to be added to the bottom of the specified geometric tolerance.

Note

If the additional text extends over multiple lines, the input string must contain `\n` characters to indicate line breaks. The output string also contains `\n` characters indicating line breaks. The text added to the top of a gtol cannot extend beyond the length of the geometric tolerance control frame.

Geometric Tolerance Text Style

The functions described in this section access the text style properties of a geometric tolerance.

Functions Introduced:

- **ProGtolTextstyleGet()**
- **ProGtolTextstyleSet()**
- **ProGtoltextTextstyleGet()**
- **ProGtoltextTextstyleSet()**
- **ProGtolTopTextHorizJustificationSet()**
- **ProGtolTopTextHorizJustificationGet()**
- **ProGtolBottomTextHorizJustificationSet()**
- **ProGtolBottomTextHorizJustificationGet()**

The function `ProGtolTextstyleGet()` returns the text style assigned of a specified geometric tolerance.

The function `ProGtolTextstyleSet()` assigns the text style of a specified geometric tolerance.

The function `ProGtoltextTextstyleGet()` retrieves the text style of the additional text applied to the specified geometric tolerance.

The function `ProGtoltextTextstyleSet()` assigns the text style of the additional text applied to the specified geometric tolerance. Specify the instance of the additional text to be accessed using the enumerated data type `ProGtolTextType`.

The function `ProGtolTopTextHorizJustificationSet()` sets the horizontal justification for the additional text applied to the specified geometric tolerance at the top.

The function `ProGtolTopTextHorizJustificationGet()` retrieves the horizontal justification for the additional text applied to the specified geometric tolerance at the top.

The function `ProGtolBottomTextHorizJustificationSet()` sets the horizontal justification for the additional text applied to the specified geometric tolerance at the bottom.

The function `ProGtolBottomTextHorizJustificationGet()` retrieves the horizontal justification for the additional text applied to the specified geometric tolerance at the bottom.

Prefix and Suffix for Geometric Tolerances

You can easily add a prefix and suffix to a geometric tolerance in both drawing and model modes. They have the same text style as the geometric tolerance text.

A prefix will be placed before the tolerance value; a suffix will be placed after the material condition, if one exists.

Functions Introduced:

- **ProGtolPrefixGet()**
- **ProGtolPrefixSet()**
- **ProGtolSuffixGet()**
- **ProGtolSuffixSet()**

The function `ProGtolPrefixGet()` obtains the prefix text for the specified geometric tolerance.

The function `ProGtolPrefixSet()` assigns the prefix set for the specified geometric tolerance.

The function `ProGtolSuffixGet()` obtains the suffix text for the specified geometric tolerance.

The function `ProGtolSuffixSet()` assigns the suffix text for the specified geometric tolerance.

Parameters for Geometric Tolerance Attributes

System parameters are automatically generated for the attributes of a geometric tolerance upon the creation of the geometric tolerance. These parameters are used for downstream processes such as Coordinate Measuring Machine (CMM) operations, and in driving other annotation and feature relationships.

Note

Parameters are generated only for geometric tolerances created within annotation features because 2D or 3D geometric tolerances created outside annotation elements do not have a placeholder for parameters.

The following table lists the various system parameters and the gtol attributes for which the parameters are generated:

Parameter	Gtol Attribute
PTC_GTOL_PRIMAY_TOL	Primary tolerance value
PTC_GTOL_TYPE	Geometric tolerance type
PTC_GTOL_MATERIAL_CONDITION	Material condition for the primary tolerance value
PTC_GTOL_COMPOSITE_TOL	Composite tolerance value; available only if the gtol type is Surface or Position
PTC_GTOL_PERUNIT_TOL	Pre-unit tolerance value; available only if the gtol type is Straightness, Perpendicular, Surface, Parallel, or Flatness
PTC_GTOL_UNITLENGTH_TOL	Unit length value; available only if the gtol type is Straightness, Perpendicular, Surface, or Parallel
PTC_GTOL_UNITAREA_TOL	Unit area value; available only if the gtol type is Flatness
PTC_GTOL_PROJECTEDTOLZONE_TOL	Projected tolerance zone value; available only if the gtol type is Angular, Perpendicular, Parallel, or Position
PTC_GTOL_UNEQUALLYDISPOSED_TOL	Unequally disposed tolerance value; available only if the gtol type is Surface

You can access the parameters of gtol attributes by using the Creo Parametric TOOLKIT parameter functions `ProParameter*Get()` and `ProParameter*Set()`, and the UDF placement functions related to variable parameters `ProUdfvarparam*()` and `ProUdfdataVarparamAdd()`. Refer to the chapter Core: Parameters on page 210 for more information on parameters.

26

Annotations: Designated Area Feature

This chapter describes how to access designated area features through Creo Parametric TOOLKIT.

Introduction to Designated Area Feature

A Designated Area is a “cosmetic surface" that can be referenced by annotations (including driven dimensions), and propagates as you add geometry to the model. It is used to indicate an area that needs to be closely examined or treated differently.

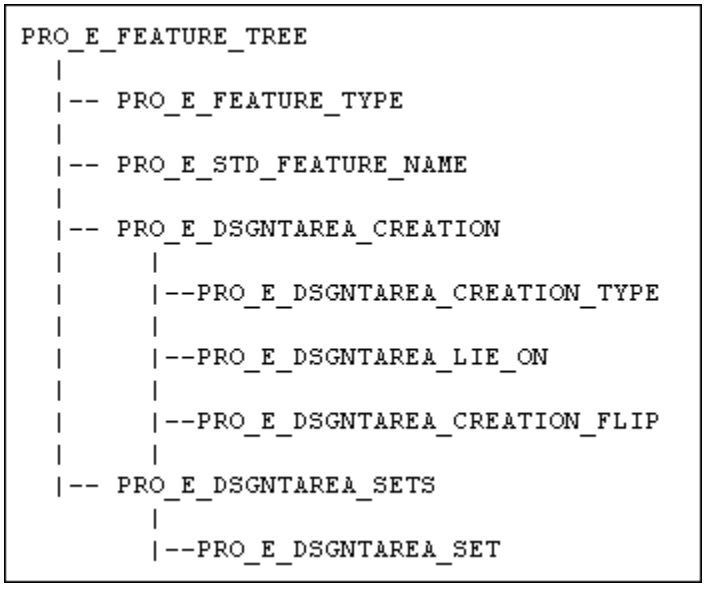
The designated area is made up of sets of chains constructed by a selection of edges or curves. The curves might lie on a solid (by default), a quilt, or on a plane. If the chains lie on multiple object types, then you must decide on one object type to place the target area.

You can attach an annotation to the created surface or to its boundaries. You can also include the designated area as a reference in a data sharing feature, if its parent surface is included. Geometry of this feature can be accessed using standard Creo Parametric TOOLKIT functions such as `ProFeatureGeomitemVisit()` and `ProSolidQuiltVisit()`.

Feature Element Tree for the Designated Area

The element tree for the Designated Area feature is documented in the header file `ProDesignatedArea.h`. The following figure demonstrates the feature element tree structure:

Feature Element Tree for a Designated Area



The following list details special information about the elements in this tree:

- PRO_E_FEATURE_TYPE—Specifies the feature type, should always have the value PRO_FEAT_DSGNT_AREA.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the designated area feature.
- PRO_E_DSGNTAREA_CREATION—Specifies the data used for creation of the designated area. It consists of the following elements:
  - PRO_E_DSGNTAREA_CREATION_TYPE—Specifies whether the designated area lies on a solid, a quilt, or on a plane, where the new surface will be constructed from a chain not related to existing surfaces. It can have one of the following values:
  - PRO_DSGNTA_CR_SOLID — Specifies that the feature is created on a solid model.
  - PRO_DSGNTA_CR_QUILT — Specifies that the feature is created on a selected quilt
  - PRO_DSGNTA_CR_AIR — Specifies that the feature is created on a plane from a chain of curves not lying on any model surface
  - PRO_E_DSGNTAREA_LIE_ON — Specifies the placement reference of the feature. If the type is PRO_DSGNTA_CR_QUILT, the feature must include the placement quilt. If the type is PRO_DSGNTA_CR_SOLID, it can optionally include a single surface of the solid, which will be the only surface of the solid used in constructing the feature. If you do not specify a single surface, all solid surfaces will be included in the designated area feature references.
  - PRO_E_DSGNTAREA_CREATION_FLIP — Specifies the flip option to switch between inside and outside the boundary chains, and can be set to either TRUE or FALSE.
- PRO_E_DSGNTAREA_SETS—Specifies an array of compound elements of the type PRO_E_DSGNTAREA_SET that contain the designated area boundaries.

Element Details of PRO_E_DSGNTAREA_SET

PRO_E_DSGNTAREA_SETS

```
|-- PRO_E_DSGNTAREA_SETS
      |
      |--PRO_E_DSGNTAREA_SET
           |
           |--PRO_E_DSGNTAREA_SET_REFS
                |
                |--PRO_E_STD_CHAIN_HOLDER
                     |
                     |--PRO_E_STD_CURVE_COLLECTION_APPL
```

Each `PRO_E_DSGNTAREA_SET` contains the following element:

`PRO_E_DSGNTAREA_SET_REFS` — Specifies the set of references to be used for creation of the designated area and consists of the following element:

- `PRO_E_STD_CHAIN_HOLDER` — Specifies the set of chains and can consists of one or more of the following element:
  - `PRO_E_STD_CURVE_COLLECTION_APPL` — Specifies the collection of curves to be used as reference. Each curve set must consist of a closed loop. For more information about curve collections, refer to the chapter, User Interface: Curve and Surface Collection.

Accessing Designated Area Properties

The following functions determine the appearance of the designated area by controlling its boundary properties such as the line style and the appearance of the surface.

Functions Introduced:

- **ProDesignatedareaStatusGet()**
- **ProDesignatedareaLinestyleGet()**
- **ProDesignatedareaLinestyleSet()**
- **ProDesignatedareaColorGet()**
- **ProDesignatedareaColorSet()**

The function `ProDesignatedareaStatusGet()` identifies the current status of the geometry of the created designated area. The input value to this function is a quilt from the designated area, or from a data sharing feature referencing the designated area.

The output geometry can have one of the following values:

- PRO_DSGNTAREA_STATUS_ACTIVE—Specifies that the geometry is active in the indicated model.
- PRO_DSGNTAREA_STATUS_INACTIVE—Specifies that the geometry is inactive due to geometry features occurring after the designated area feature in the regeneration order.
- PRO_DSGNTAREA_STATUS_OUT_OF_COPIED_GEOM—Specifies that the parent geometry used from the trimming was cut out from the model.

The function ProDesignatedareaLinestyleGet() returns the line style used for the boundary of the designated area. Use the function ProDesignatedareaLinestyleSet() to set the line style.

The function ProDesignatedareaColorGet() returns the color used for the boundary of the designated area. Use the function ProDesignatedareaColorSet() to set the color.

27

Data Management: Windchill Operations

Creo Parametric has the capability to be directly connected to Windchill solutions, including Windchill ProjectLink, Pro/INTRALINK and PDMLink servers. This access allows users to manage and control the product data seamlessly from within Creo Parametric.

This chapter lists Creo Parametric TOOLKIT APIs that support Windchill servers and server operations in a connected Creo Parametric session.

Introduction

The functions introduced in this chapter provide support for the basic Windchill server operations from within Creo Parametric. Refer to the compatibility matrix on PTC.com for information on the versions of Windchill compatible with Creo Parametric.

With these functions, operations such as registering a Windchill server, managing workspaces, and check in or check out of objects will be possible via Creo Parametric TOOLKIT.

The capabilities of the APIs described in this chapter are similar to the operations available from within the Creo Parametric client, with some restrictions. Some of the APIs specified in this section are supported in non-interactive mode, that is, batch mode application or asynchronous application.

Note

When Creo Parametric applications are running in asynchronous non-graphical mode, they require login credentials before execution. If you want to override the requirement of specifying login credentials for Creo Parametric applications, set the environment variable `PROWT_AUTH_MODE` to `PROWT_AUTH_UNATTENDED`.

For more information about batch mode refer to the section Using Creo Parametric TOOLKIT to Make a Batch Creo Parametric Session on page 52 and for asynchronous mode refer to the chapter Core: Asynchronous Mode on page 277.

Accessing a Windchill Server from a Creo Parametric Session

Creo Parametric allows you to register Windchill servers as a connection between the Windchill database and Creo Parametric. Although the represented Windchill database can be from ProjectLink, Pro/INTRALINK or PDMLink, all types of databases are represented in the same way.

You can use the following identifiers when referring to Windchill servers in Creo Parametric TOOLKIT:

- `Codebase URL`—This is the root portion of the URL that is used to connect to a Windchill server. For example, http://wcserver.company.com/Windchill.
- `Server Alias`—A server alias is used to refer to the server after it has been registered. The alias is also used to construct paths to files in the server workspaces and commonspaces. The server alias is chosen by the user or

application and it need not have any direct relationship to the codebase URL. An alias can be any normal name, such as `my_alias`.

Accessing Information Before Registering a Server

To start working with a Windchill server, you must establish a connection by registering the server in Creo Parametric. The functions described in this section enable you to connect to a Windchill server and access information related to the server.

Functions Introduced:

- **ProBrowserAuthenticate()**
- **ProServerClassGet()**
- **ProServerVersionGet()**
- **ProServerContextsCollect()**
- **ProServerWorkspacesCollect()**

Use the function `ProBrowserAuthenticate()` to set the authentication context using a valid username and password. A successful call to this function allows the Creo Parametric session to register with any server accepting the username and password combination. A successful call to this function also ensures that an authentication dialog box does not appear during the registration process. You can call this function any number of times to set the authentication context for any number of Windchill servers, provided that you register the appropriate servers or servers immediately after setting the context.

The function `ProServerClassGet()` returns the class of the server. The values are:

- `Windchill`—Denotes a WindchillPDMLink server.
- `ProjectLink`—Denotes WindchillProjectLink type of servers.

This function accepts the server codebase URL as the input.

The function `ProServerVersionGet()` returns the version of Windchill that is configured on the server, for example, `9.0` or `10.0`. This function accepts the server codebase URL as the input.

Note

`ProServerVersionGet()` works only for Windchill servers and returns the `PRO_TK_UNSUPPORTED` error, if the server is not a Windchill server.

The function `ProServerContextsCollect()` gives a list of all the available contexts for a specified server. A context is used to associate a workspace with a product, project, or library. This function accepts the server codebase URL as the input.

The function `ProServerWorkspacesCollect()` returns the list of available workspaces for the specified server. The workspace data returned contains the name of the workspace and its context. This function accepts the server codebase URL as the input.

Registering and Activating a Server

From Creo Parametric 2.0 onward, the Creo Parametric TOOLKIT APIs call the same underlying API as Creo Parametric to register and unregister servers. Hence, registering the servers using Creo Parametric TOOLKIT APIs is similar to registering the servers using the Creo Parametric user interface. Therefore, the servers registered by Creo Parametric TOOLKIT are available in the Creo Parametric Server Registry. The servers are also available in other locations in the Creo Parametric user interface such as, the **Folder Navigator** and the embedded browser.

Functions Introduced:

- **ProServerRegister()**
- **ProServerActivate()**
- **ProServerUnregister()**

The function `ProServerRegister()` registers the specified server with the codebase URL. You can automate the registration of servers in interactive mode. To preregister the servers use the standard `config.fld` setup. If you do not want the servers to be preregistered in batch mode, set the environment variable `PTC_WF_ROOT` to an empty directory before starting Creo Parametric.

A successful call to `ProBrowserAuthenticate()` with a valid username and password is essential for `ProServerRegister()` to register the server without launching the authentication dialog box. Registration of the server establishes the server alias.

The function `ProServerActivate()` sets the specified server as the active server in the Creo Parametric session.

The function `ProServerUnregister()` unregisters the specified server. This is similar to **Server Registry ► Delete** through the user interface.

Accessing Information From a Registered Server

Functions Introduced:

- **ProServerActiveGet()**
- **ProServerContextGet()**
- **ProServerAliasGet()**
- **ProServersCollect()**
- **ProServerLocationGet()**
- **ProServerIsOnline()**

The function `ProServerActiveGet()` returns the primary server.

The function `ProServerContextGet()` returns the active context of the active server.

The function `ProServerAliasGet()` returns the alias of a server if you specify the codebase URL.

The function `ProServersCollect()` returns a list of the aliases of all the registered servers.

The function `ProServerLocationGet()` returns the codebase URL for the server if you specify the alias.

The function `ProServerIsOnline()` checks if the specified server is online or offline. It returns `PRO_B_TRUE` if the server is online.

Accessing the Workspace

For every workspace, a new distinct storage location is maintained in the user's personal folder on the server (server-side workspace) and on the client (client-side workspace cache). Together, the server-side workspace and the client-side workspace cache make up the workspace.

Workspace Data

Functions Introduced:

- **ProServerworkspacedataAlloc()**
- **ProServerworkspacedataNameGet()**
- **ProServerworkspacedataContextGet()**
- **ProServerworkspacedataFree()**
- **ProServerworkspacedataProarrayFree()**

The workspace data is an opaque handle that contains the name and context of the workspace. The function `ProServerWorkspacesCollect()` returns an array of workspace data. Workspace data is also required for the function `ProServerWorkspaceCreate()` to create a workspace with a given name and a specific context.

The function `ProServerworkspacedataAlloc()` creates a workspace data structure to describe a workspace. The workspace contains the name of the workspace and the context in which the workspace is stored.

The function `ProServerworkspacedataNameGet()` retrieves the name of the workspace from the workspace data.

The function `ProServerworkspacedataContextGet()` retrieves the context of the workspace from the workspace data.

Use the function `ProServerworkspacedataFree()` to free the workspace data structure from memory.

Use the function `ProServerworkspacedataProarrayFree()` to free the workspace data array from the memory.

Creating and Modifying the Workspace

Functions Introduced

- **ProServerWorkspaceCreate()**
- **ProServerWorkspaceGet()**
- **ProServerWorkspaceSet()**
- **ProServerWorkspaceDelete()**

The function `ProServerWorkspaceCreate()` creates and activates a new workspace.

The function `ProServerWorkspaceGet()` retrieves the name of the active workspace.

The function `ProServerWorkspaceSet()` sets a specified workspace as an active workspace.

The function `ProServerWorkspaceDelete()` deletes the specified workspace. The function deletes the workspace only if the following conditions are met:

- The workspace is not the active workspace.
- The workspace does not contain any checked out objects.

Use one of the following techniques to delete an active workspace:

- Make the required workspace inactive using `ProServerWorkspaceSet()` with the name of some other workspace and then call `ProServerWorkspaceDelete()`.
- Unregister the server using `ProServerUnregister()` and delete the workspace using the codebase URL instead of the alias.

Workflow to Register a Server

To Register a Server with an Existing Workspace

Perform the following steps to register a Windchill server with an existing workspace:

1. Set the appropriate authentication context using the function `ProBrowserAuthenticate()` with a valid username and password.
2. Look up the list of workspaces using the function `ProServerWorkspacesCollect()`. If you already know the name of the workspace on the server, then ignore this step.
3. Register the workspace using the function `ProServerRegister()` with an existing workspace name on the server.
4. Activate the server using the function `ProServerActivate()`.

To Register a Server with a New Workspace

Perform the following steps to register a Windchill server with a new workspace:

1. Perform steps 1 to 4 in the preceding section to register the Windchill server with an existing workspace.
2. Use the function `ProServerContextsCollect()` to choose the required context for the server.
3. Create a new workspace with the required context using the function `ProServerWorkspaceCreate()`. This function automatically makes the created workspace active.

Note

You can create a workspace only after the server is registered.

Aliased URL

An aliased URL serves as a handle to the server objects. You can access the server objects in the commonspace (shared folders) and the workspace using the aliased URL. An aliased URL is a unique identifier for the server object and the format is as follows:

- Object in workspace has a prefix `wtws`

`wtws://<server_alias>/<workspace_name>/<object_server_name>`

where `<object_server_name>` includes `<object_name>.<object_extension>`

For example, `wtws://my_server/my_workspace/abcd.prt`, `wtws://my_server/my_workspace/intf_file.igs`

where

`<server_alias>` is `my_server`

`<workspace_name>` is `my_workspace`

- Object in commonspace has a prefix `wtpub`

  `wtpub://<server_alias>/<folder_location>/<object_server_name>`

  For example, `wtpub://my_server/path/to/cs_folder/abcd.prt`

  where

  `<server_alias>` = `my_server`

  `<folder_location>` = `path/to/cs_folder`

Note

- `<object_server_name>` must be in lowercase.
- The APIs are case-sensitive to the aliased URL.
- `<object_extension>` should not contain Creo Parametric versions, for example, .1 or .2, and so on.

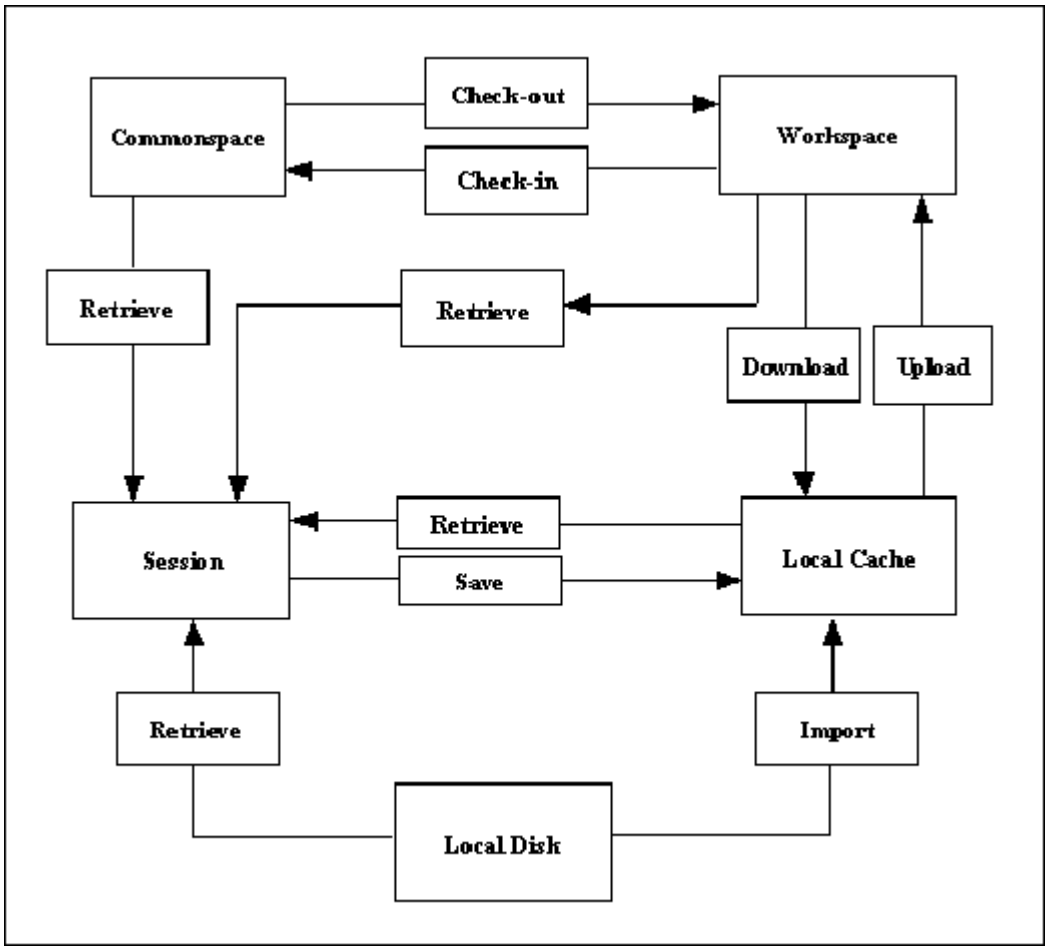
Server Operations

After registering the Windchill server with Creo Parametric, you can start accessing the data on the Windchill servers. The Creo Parametric interaction with Windchill servers leverages the following locations:

- Commonspace (Shared folders)
- Workspace (Server-side workspace)
- Workspace local cache (Client-side workspace)

- Creo Parametric session
- Local disk

The functions described in this section enable you to perform the basic server operations. The following diagram illustrates how data is transferred among these locations.



Save

Functions Introduced:

- **ProMdlSave()**

The function `ProMdlSave()` stores the object from the session in the local workspace cache, when a server is active.

Upload

An upload transfers Creo Parametric files and any other dependencies from the local workspace cache to the server-side workspace.

Function Introduced:

- **ProServerObjectsUpload()**

The function `ProServerObjectsUpload()` uploads the specified object to the workspace. The object to be uploaded must be present in the current Creo Parametric session. Additionally, ensure that you save the object using the function `ProMdlSave()` before you upload it.

Note

To upload all the objects to the workspace without retrieving them in the current Creo Parametric session, use the function `ProServerObjectsCheckin()` with the checkin option *upload_only* set to `PRO_B_TRUE`.

CheckIn

After you have finished working on objects in your workspace, you can share the design changes with other users. The checkin operation copies the information and files associated with all changed objects from the workspace to the Windchill database.

Functions Introduced

- **ProServerObjectsCheckin()**
- **ProServercheckinoptsAlloc()**
- **ProServercheckinoptsDeflocationSet()**
- **ProServercheckinoptsLocationAdd()**
- **ProServercheckinoptsBaselineSet()**
- **ProServercheckinoptsKeepcheckedoutSet()**
- **ProServercheckinoptsAutoresolveSet()**
- **ProServercheckinoptsUploadonlySet()**
- **ProServercheckinoptsFree()**

The function ProServerObjectsCheckin() checks in or uploads an object to the database. The object to be checked in or uploaded must be present in the current Creo Parametric session. Changes made to the object are not included unless you save the object to the workspace using the function ProMdlSave() before it is checked in or uploaded.

Note

ProServerObjectsCheckin() checks in the target object by default. To only upload the object, set the checkin option *upload_only* to PRO_B_TRUE.

If you pass NULL as the value of the input argument *options*, the checkin operation is similar to the **Auto Check In** option in Creo Parametric. For more details on **Auto Check In**, refer to the online help for Creo Parametric.

By using an appropriately constructed *options* argument, you can control the checkin or upload operation. The APIs described in this section help in constructing the *options* argument.

The function ProServercheckinoptsAlloc() allocates a set of checkin or upload options for the object. These options are as follows:

- Default location—Specifies the default *folder_location* on the server for the automatic checkin operation. Use the function ProServercheckinoptsDeflocationSet() to set this location.
- Server location—Specifies the *folder_location* on the server in which an object will be checked in or uploaded. Use the function ProServercheckinoptsLocationAdd() to set this location.
- *Baseline*—Specifies the baseline information for the objects upon checkin. This information does not apply to upload operations. Use the function ProServercheckinoptsBaselineSet() to create a new baseline. The baseline information for a checkin operation is as follows:
  - *baseline_name*—Specifies the name of the baseline.
  - *baseline_number*—Specifies the number of the baseline.

  The default format for the baseline name and baseline number is Username + time (GMT) in milliseconds.
  - *baseline_location*—Specifies the location of the baseline.
  - *baseline_lifecycle*—Specifies the name of the lifecycle.
- *keep_checked_out*—If this option is set to PRO_B_TRUE, then the contents of the selected object are checked in to the Windchill server and automatically checked out again for further modification. The default value is PRO_B_FALSE. This option does not apply to upload operations. Use the function

ProServercheckinoptsKeepcheckedoutSet() to set the *keep_checked_out* flag.

- *autoresolve*—Specifies the option used to automatically resolve objects that have not been completely checked in or uploaded to the database. The autoresolve options specified by the enumerated type ProServerAutoresolveOption are as follows:
  - PRO_SERVER_DONT_AUTORESOLVE—Model references missing from the workspace are not automatically resolved. This may result in a conflict upon checkin. This option is used by default.
  - PRO_SERVER_AUTORESOLVE_IGNORE—Missing references are automatically resolved by ignoring them.
  - SERVER_AUTORESOLVE_UPDATE_IGNORE—Missing references are automatically resolved by updating them in the database and ignoring them if not found.

  Use the function ProServercheckinoptsAutoresolveSet() to assign the appropriate *autoresolve* option.
- *upload_only*—Specifies the option to fully check in the target object or only upload the object to the server. Set this option to PRO_B_TRUE to only upload and not check in the target objects and to PRO_B_FALSE to upload and check in the objects. By default, this option is PRO_B_FALSE, if not explicitly set, to cause a checkin. Use the function ProServercheckinoptsUploadonlySet() to set the *upload_only* flag.

Use the function ProServercheckinoptsFree() to free the memory of the checkin options.

Notification Functions

A Creo Parametric TOOLKIT notification is called before you attempt to check in a model through the Creo Parametric user interface. The notification functions are established in a session using the function ProNotificationSet().

Function Introduced:

- **ProCheckinUIPreAction()**

The notification function ProCheckinUIPreAction() is called when you select **File**>**Auto Check In**, **File**>**Custom Check In**, or the corresponding options on the **Model Tree** menu in the Creo Parametric user interface. It is called before any action is performed by the corresponding menu commands. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_CHECKIN_UI_PRE.

Retrieval

Standard Creo Parametric TOOLKIT provides several functions that are capable of retrieving models. When using these functions with Windchill servers, remember that these functions do not check out the object to allow modifications.

Functions Introduced:

- **ProMdlnameRetrieve()**
- **ProMdlFiletypeLoad()**

The function `ProMdlnameRetrieve()` loads an object into a session given its name and type. This function searches for the object in the active workspace, the local directory, and any other paths specified by the configuration option `search_path`.

The function `ProMdlFiletypeLoad()` loads an object into session given its path. The path can be a disk path, a workspace path, or a commonspace path. Refer to the section Aliased URL on page 646 for examples of these types of paths.

Checkout and Download

To modify an object from the commonspace, you must check out the object. The process of checking out communicates your intention to modify a design to the Windchill server. The object in the database is locked, so that other users can obtain read-only copies of the object, and are prevented from modifying the object while you have it checked out.

Checkout is often accompanied by a download action, where the objects are brought from the server-side workspace to the local workspace cache. In Creo Parametric TOOLKIT, both operations are covered by the same set of functions.

Functions Introduced:

- **ProServerObjectsCheckout()**
- **ProServerMultiobjectsCheckout()**
- **ProServercheckoutoptsAlloc()**
- **ProServercheckoutoptsFree()**
- **ProServercheckoutoptsDependencySet()**
- **ProServercheckoutoptsIncludeinstancesSet()**
- **ProServercheckoutoptsVersionSet()**
- **ProServercheckoutoptsDownloadSet()**
- **ProServercheckoutoptsReadonlySet()**

The function ProServerObjectsCheckout() checks out and optionally downloads the object to the workspace based on the configuration specifications of the workspace. It takes the following two potential identifiers for the model to checkout:

- ProMdl handle—Specifies the object to be checked out. This is applicable if the model has already been retrieved without checking out.
- Aliased URL—Specifies the commonspace path of the object.

Use the function ProServerMultiobjectsCheckout() to check out and download a ProArray of objects to the workspace based on the configuration specifications of the workspace.

Note

Creo Parametric TOOLKIT functions do not support the AS_STORED configuration.

If you specify the value of the input argument *checkout* as PRO_B_TRUE in the above functions, the selected object is checked out. Otherwise, the object is downloaded without being checked out. The download action enables you to bring read-only copies of objects into your workspace. This allows you to examine the object without placing a lock on it.

If you pass NULL as the value of the input argument *options* in the above functions, then the default Creo Parametric checkout rules apply.

The function ProServercheckoutoptsAlloc() allocates a set of checkout options for the object. These options are as follows:

- *dependency*—Specifies the dependency rule used while checking out dependents of the object selected for checkout. The function ProServercheckoutoptsDependencySet() sets the dependency rule for checkout. The types of dependencies specified by the enumerated type ProServerDependency are as follows:
  - PRO_SERVER_DEPENDENCY_ALL—All the objects that are dependent on the selected object are downloaded, that is, they are added to the workspace.
  - PRO_SERVER_DEPENDENCY_REQUIRED—All the objects that are required to successfully retrieve the selected object in the CAD application are downloaded, that is, they are added to workspace.
  - PRO_SERVER_DEPENDENCY_NONE—None of the dependent objects

from the selected object are downloaded, that is, they are not added to workspace.

- *include_option*—Specifies the rule for including instances from the family table during checkout. The function `ProServercheckoutoptsIncludeinstancesSet()` sets the flag to include instances during checkout. The type of instances specified by the enumerated type `ProServerInclude` are as follows:
  - `PRO_SERVER_INCLUDE_ALL`—All the instances of the selected object are checked out.
  - `PRO_SERVER_INCLUDE_SELECTED`—The application selects the instance members from the family table to be included during checkout.
  - `PRO_SERVER_INCLUDE_NONE`—No additional instances from the family table are added to the object list.
- *version*—Specifies the version of the object that is checked out or downloaded to the workspace. If *version* is not set, the object is checked out according to the current workspace configuration. The function `ProServercheckoutoptsVersionSet()` sets the version of the object.
- *download*—Specifies the checkout type as `download` or `link`. The value `download` specifies that the object content is downloaded and checked out, while `link` specifies that only the metadata is downloaded and checked out. Use the function `ProServercheckoutoptsDownloadSet()` to set this option.
- *readonly*—Downloads the file without checking out the file. To use this option you must set the *checkout* argument of the function `ProServerObjectsCheckout()` as `PRO_B_FALSE`. Use the function `ProServercheckoutoptsReadonlySet()` to set the *readonly* flag to `PRO_B_TRUE`.

Use the function `ProServercheckoutoptsFree()` to free the memory of the checkout options.

The following truth table explains the dependencies of the different control factors in `ProServerObjectsCheckout()` and the effect of different combinations on the end result.

Argument *checkout* in ProServerObjects Checkout()	ProServer checkoutopts DownloadSet()	ProServer checkoutopts ReadonlySet()	Result
`PRO_B_TRUE`	`PRO_B_TRUE`	NA	Object is checked out and its content is downloaded.
`PRO_B_TRUE`	`PRO_B_FALSE`	NA	Object is checked out but content is not downloaded.

Argument *checkout* in ProServerObjects Checkout()	ProServer checkoutopts DownloadSet()	ProServer checkoutopts ReadonlySet()	Result
PRO_B_FALSE	NA	PRO_B_TRUE	Object is downloaded without checkout and as read-only.
PRO_B_FALSE	NA	PRO_B_FALSE	This combination is invalid and is not supported.

The function ProServercheckoutoptsReadonlySet() can be used to download objects without checking them out. To download objects, you must set the *checkout* argument of the function ProServerObjectsCheckout() as PRO_B_FALSE before using ProServercheckoutoptsReadonlySet().

The following table describes the different values that can be specified for the check out options for the functions ProServercheckoutoptsReadonlySet() and ProServerObjectsCheckout(). It also describes the actions that can be performed on the downloaded or checked out object from the Creo Parametric user interface or using the Creo Parametric TOOLKIT applications.

The following table explains the dependencies of the different control factors on ProServercheckoutoptsReadonlySet():

Argument *readonly* in ProServer-checkoutopts-ReadonlySet()	Argument *checkout* in ProServerOb-jectsCheck-out()	State of Object Content	Options Available in the Creo Parametric User Interface	Actions that can be Performed Using Creo Parametric TOOLKIT Applications
PRO_B_TRUE	PRO_B_FALSE	Downloaded but not checked out	The **Conflicts** dialog box allows you to perform one of the following operations on the object: check out, revise and check out, continue with modifications, or make the object read only. Refer to the Creo Parametric Help for more information on resolving conflicts.	Check out the object and modify it
PRO_B_FALSE	PRO_B_FALSE	Checked out	The object can be modified.	The object can be modified.

Argument *readonly* in ProServer-checkoutopts-ReadonlySet()	Argument *checkout* in ProServerOb-jectsCheck-out()	State of Object Content	Options Available in the Creo Parametric User Interface	Actions that can be Performed Using Creo Parametric TOOLKIT Applications
PRO_B_TRUE	PRO_B_TRUE	Checked out	The object can be modified.	The object can be modified.
No option set	PRO_B_FALSE	Downloaded but not checked out	The **Conflicts** dialog box allows you to perform one of the following operations on the object: check out, revise and check out, continue with modifications, or make the object read only.	Check out the object and modify it

Undo Checkout

Function Introduced:

- **ProServerObjectsUndocheckout()**

Use the function `ProServerObjectsUndocheckout()` to undo a checkout of the specified object. When you undo a checkout, the changes that you have made to the content and metadata of the object are discarded and the content, as stored in the server, is downloaded to the workspace. This function is applicable only for the model in the active Creo Parametric session.

Import and Export

Creo Parametric TOOLKIT provides you with the capability of transferring specified objects to and from a workspace. Import and export operations must take place in a session with no models.

Functions Introduced:

- **ProCurrentWorkspaceImport()**
- **ProCurrentWorkspaceExport()**
- **ProCurrentWorkspaceImpexMessagesGet()**
- **ProWsimpexmessageDataGet()**
- **ProWsimpexmessageArrayFree()**
- **ProWsexportSecondarycontentoptionSet()**

The function `ProCurrentWorkspaceImport()` imports specified objects from disk to the current workspace in a linked session of Creo Parametric.

The function `ProCurrentWorkspaceExport()` exports the specified objects from the current workspace to a location on disk in a linked session of Creo Parametric.

Both `ProCurrentWorkspaceImport()` and `ProCurrentWorkspaceExport()` allow you to specify a dependency criterion to process the following items:

- All external dependencies
- Only required dependencies
- No external dependencies

All warnings, conflicts, or errors generated during import or export operations are logged in the `proimpex.errors` file created in the Creo Parametric working directory. Alternatively, you can obtain this information using the function `ProCurrentWorkspaceImpexMessagesGet()`. This function returns a `ProArray` of messages generated by the last call to `ProCurrentWorkspaceImport()` or `ProCurrentWorkspaceExport()`.

The function `ProWsimpexmessageDataGet()` extracts the contents of the message generated by `ProCurrentWorkspaceImport()` or `ProCurrentWorkspaceExport()`. The message contains the following items:

- *type*—Specifies the severity of the message in the form of the enumerated type `ProWSImpexMessageType`. The severity is one of the following types:
  - `PRO_WSIMPEX_MSG_INFO`—Specifies an informational type of message.
  - `PRO_WSIMPEX_MSG_WARNING`—Specifies a low severity problem that can be resolved according to the configured rules.
  - `PRO_WSIMPEX_MSG_CONFLICT`—Specifies a conflict that can be overridden.
  - `PRO_WSIMPEX_MSG_ERROR`—Specifies a conflict that cannot be overridden or a serious problem that prevents processing of an object.
- *object*—Specifies the object name or the name of the object path described in the message.
- *description*—Specifies the description of the problem or the message information.
- *resolution*—Specifies the resolution applied to resolve a conflict that can be

overridden. This is applicable when the message is of the type PRO_WSIMPEX_MSG_CONFLICT.

- *succeeded*—Determines whether the resolution succeeded or not. This is applicable when the message is of the type PRO_WSIMPEX_MSG_CONFLICT.

Use the function ProWsimpexmessageArrayFree() to free the memory allocated to the array of messages returned by ProCurrentWorkspaceImpexMessagesGet().

The function ProWsexportSecondarycontentoptionSet() sets the ProBoolean option that controls the export of secondary contents. If this option is set to PRO_B_TRUE, secondary contents are exported along with the primary Creo Parametric model files. By default, it is PRO_B_TRUE.

File Copy

Creo Parametric TOOLKIT provides you with the capability of copying a file from the workspace or target folder to a location on the disk and vice-versa.

Functions Introduced:

- **ProFileCopyToWS()**
- **ProFileCopyFromWS()**
- **ProFileCopyFromWSDocument()**
- **ProFileselectionDocNameGet()**
- **ProDocumentFileContentsRead()**

Use the function ProFileCopyToWS() to copy a file from disk to the workspace. The file can optionally be added as secondary content to a given workspace file. If the viewable file is added as secondary content, a dependency is created between the Creo Parametric model and the viewable file.

Use the function ProFileCopyFromWS() to copy a file from the workspace to a location on disk.

Note

When importing or exporting Creo Parametric models, PTC recommends that you use `ProCurrentWorkspaceImport()` and `ProCurrentWorkspaceExport()`, respectively, to perform the operation. Functions that copy individual files do not traverse Creo Parametric model dependencies, and therefore do not copy a fully retrievable set of models at the same time.

Additionally, only the functions `ProCurrentWorkspaceImport()` and `ProCurrentWorkspaceExport()` provide full metadata exchange and support. That means `ProCurrentWorkspaceImport()` can communicate all the Creo Parametric designated parameters, dependencies, and family table information to a PDM system while `ProCurrentWorkspaceExport()` can update exported Creo Parametric data with PDM system changes to designated and system parameters, dependencies, and family table information. Hence PTC recommends the use of `ProFileCopyToWS()` and `ProFileCopyFromWS()` to process only non-Creo Parametric files.

The function `ProFileCopyFromWSDocument()` copies a primary or secondary file from the workspace to the specified location on disk. The input arguments are:

- *source_file*—Specifies the path to the primary or secondary file. The path must be specified as `wtws://<path to the file>`.

  Use the functions such as, `ProFileMdlnameOpen()` and `ProFileMdlfiletypeOpen()`, to get the path to the files.
- *document_name*—Specifies the name of the primary file, which is associated with the secondary file specified in the argument *source_file*. Use the function `ProFileselectionDocNameGet()` to get the name of the primary file for a secondary file.

  In the argument *source_file*, if a primary file is specified, then pass the argument *document_name* as `NULL`.
- *target_directory*—Specifies a path on the local disk where the file must be copied.

The function `ProFileselectionDocNameGet()` returns the name of the primary file for the selected secondary file. The secondary files are selected in the file open functions. The functions such as, `ProFileOpen()`, `ProFileMdlnameOpen()`, `ProFileMdlfiletypeOpen()` and so on, are used to open the dialog box where you can browse and select a secondary file. The function `ProFileselectionDocNameGet()` returns the name of the

primary file for the last selected secondary file in the file open functions. If you select a primary file in these file open functions, then the function `ProFileselectionDocNameGet()` returns the error `PRO_TK_E_NOT_FOUND`.

The function `ProDocumentFileContentsRead()` reads the contents of the specified file. The file can be located on the local disk or Windchill. The function returns a `ProArray` of characters. Declare the output variable as `char*` and typecast it as `ProArray*` when you pass it to the API. Use the function `ProArrayFree()` to free the `ProArray`.

Note

The function `ProDocumentFileContentsRead()` is not supported for CAD models.

Server Object Status

Function Introduced:

- **ProServerObjectIsModified()**

The function `ProServerObjectIsModified()` verifies the current status of the object in the workspace as well as in the local workspace cache. The status of the object is as follows:

- *checkout_status*—Specifies whether the object is checked out for modification. The value `PRO_B_TRUE` indicates that the specified object is checked out to the active workspace.

  The value `PRO_B_FALSE` indicates one of the following statuses:

  - The specified object is not checked out
  - The specified object is only uploaded to the workspace, but was never checked in
  - The specified object is only saved to the local workspace cache, but was never uploaded
- *modifiedInWS*—Specifies whether the object has been modified in the workspace since checkout. The value of this argument is `PRO_B_FALSE` if the newly created object has not been uploaded.
- *modifiedLocally*—Specifies whether the object has been modified in the local workspace cache. The value of this argument is `PRO_B_TRUE` if the object has been saved in the local workspace cache. The argument returns `PRO_B_`

FALSE if the object has not been saved after modifying it in the local workspace cache.

Note

The function ProServerObjectStatusGet() is deprecated. Use the function ProServerObjectIsModified() instead.

Delete Objects

Function Introduced:

- **ProServerObjectsRemove()**

The function ProServerObjectsRemove() deletes the array of objects from the workspace. When passed with the *mode_names* array as NULL, this function removes all the objects in the active workspace.

Conflicts During Server Operations

Functions Introduced:

- **ProServerconflictsDescriptionGet()**
- **ProServerconflictsFree()**

Conflict objects are provided to capture the error condition while performing the following server operations using the specified APIs:

Operation	API	Error Object
Checkin an object or workspace	ProServerObjectsCheckin()	ProServerCheckinConflicts
Checkout an object	ProServerObjectsCheckout()	ProServerCheckoutConflicts
Undo checkout of an object	ProServerObjectsUndocheckout()	ProServerUndoCheckoutConflicts
Upload object	ProServerObjectsUpload()	ProServerUploadConflicts
Download object	ProServerObjectsCheckout() (with download as PRO_B_TRUE)	ProServerCheckoutConflicts
Delete workspace	ProServerWorkspaceDelete()	ProServerDeleteConflicts
Remove object	ProServerObjectsRemove()	ProServerRemoveConflicts

These APIs return a common status PRO_TK_CHECKOUT_CONFLICT and a conflict object ProServerConflicts. The conflict object is used to get more details about the error condition.

Use the function `ProServerConflictsDescriptionGet()` to extract details of the error condition. This description is similar to the description displayed by the Creo Parametric HTML User Interface in the conflict report.

The function `ProServerconflictsFree()` frees the memory of the conflict structure returned by the functions.

Utility APIs

The functions specified in this section enable you to obtain the handle to the server objects to access them. The handle may be the Aliased URL or the model name of the http URL. These utilities enable the conversion of one type of handle to the other type.

Functions Introduced:

- **ProServerModelNameToAliasedURL()**
- **ProServerAliasedURLToModelName()**
- **ProServerAliasedURLToURL()**

The function `ProServerModelNameToAliasedURL()` enables you to search for a server object by its name. Specify the complete file name of the object as the input, for example, `test_part.prt`. The function returns the aliased URL for a model on the server. For more information regarding the aliased URL, refer to the section Aliased URL on page 646. During the search operation, the workspace takes precedence over the shared space.

You can also use this function to search for files that are not in the Creo Parametric format. For example, `my_text.txt`, `prodev.dat`, `intf_file.stp`, and so on.

The function `ProServerAliasedURLToModelName()` returns the name of the object from the given aliased URL on the server.

The function `ProServerAliaseURLToURL()` converts an aliased URL to a standard URL to the objects on the server.

For example, `wtws://my_alias/Wildfire/abcd.prt` is converted to an appropriate URL on the server as http://server.mycompany.com/Windchill.

Sample Batch Workflow

A typical workflow using the Windchill APIs for an asynchronous non-graphical application is as follows:

1. Start a Creo Parametric session using the function `ProEngineerConnectionStart()`.
2. Authenticate the browser using the function `ProBrowserAuthenticate()`.
3. Register the server with the new workspace using the function `ProServerRegister()`.
4. Activate the server using the function `ProServerActivate()`.
5. Check out and retrieve the model from the vault URL using the function `ProServerObjectsCheckout()` followed by `ProMdlnameRetrieve()`.
6. Modify the model according to the application logic.
7. Save the model to the workspace using the function `ProMdlSave()`.
8. Check in the modified model back to the server using the function `ProServerObjectsCheckin()`.
9. After processing all models, unregister from the server using the function `ProServerUnregister()`.
10. Delete the workspace using `ProServerWorkspaceDelete()`.
11. Stop Creo Parametric using the function `ProEngineerEnd()`.

28

Interface: Data Exchange

This chapter describes various methods of importing and exporting files in Creo Parametric TOOLKIT.

Exporting Information Files

Functions Introduced:

- **ProOutputFileMdlnameWrite()**

The function `ProOutputFileMdlnameWrite()` is used to create files of several types from data in Creo Parametric. This function operates only on the current object. The file types are declared in `ProUtil.h`. The export formats and their type constants are as listed in the following table.

Export Format	Creo Parametric TOOLKIT function	Type Constant
Bills of material	`ProOutputFileMdlname Write()`	`PRO_BOM_FILE`
Drawing setup file		`PRO_DWG_SETUP_FILE`
Feature identifier		`PRO_FEAT_INFO`, `PRO_FEAT_ INFO`
Material file (currently assigned material)		`PRO_MATERIAL_FILE`
CL Data output, NC Sequence file		`PRO_MFG_FEAT_CL`
CL Data operation file		`PRO_MFG_OPER_CL`
Information on Creo Parametric Objects		`PRO_MODEL_INFO`
Program file		`PRO_PROGRAM_FILE`
Cable Parameters file		`PRO_CABLE_PARAMS_FILE`
Connector Parameters file		`PRO_CONNECTOR_PARAMS_FILE`
Spool file		`PRO_SPOOL_FILE`
Difference Report file		`PRO_DIFF_REPORT_FILE`
IGES file		`PRO_IGES_FILE`
DXF file		`PRO_DXF_FILE`
DWG file		`PRO_DWG_FILE`
Render file		`PRO_RENDER_FILE`
SLA ASCII file		`PRO_SLA_ASCII_FILE`
SLA Binary file		`PRO_SLA_BINARY_FILE`
INVENTOR file		`PRO_INVENTOR_FILE`
CATIA facets file		`PRO_CATIAFACETS_FILE`
IGES 3D file		`PRO_IGES_3D_FILE`
STEP file		`PRO_STEP_FILE`
VDA file		`PRO_VDA_FILE`
FIAT file		`PRO_FIAT_FILE`
CATIA DIRECT file		`PRO_CATIA_DIRECT_FILE`
ACIS file		`PRO_ACIS_FILE`
CGM file		`PRO_CGM_FILE`

The option `PRO_RELATION_FILE` creates a file that contains a list of all the model relations and parameters.

To access parameters on the connector entry ports, you must call the function`ProOutputFileMdlnameWrite()` with the option `PRO_CONNECTOR_PARAMS_FILE`. The function writes a text file to disk. This text file is in the same format as the file that you edit when using the Creo Parametric command **Connector ► Modify Parameters ► Mod Param**.

To generate and export a difference report to text or CSV format, call the function `ProOutputFileMdlnameWrite()` with the option `PRO_DIFF_REPORT_FILE`.

Note

As this report is generated and exported from the Creo Parametric embedded browser, using this output type will cause Creo Parametric to show the difference report in the browser.

For some of the options used with `ProOutputFileMdlnameWrite()`, you must provide some more information, using the last four arguments. The following list shows the arguments to be set and when:

- For `PRO_RENDER_FILE`, `PRO_INVENTOR_FILE`, `PRO_CATIAFACETS_FILE`, `PRO_SLA_ASCII_FILE`, and `PRO_SLA_BINARY_FILE`, set the following argument:
  - *arg1*—The name of the coordinate system. If this `NULL`, the function uses the default coordinate system.
- For `PRO_SPOOL_FILE`, set *arg1* to the spool name.
- For `PRO_FEAT_INFO`, `PRO_MFG_FEAT_CL`, and `PRO_MFG_OPER_CL`, set the following argument:
  - *arg2*—The integer identifier of the feature.
- For `PRO_IGES_3D_FILE`, `PRO_STEP_FILE`, `PRO_VDA_FILE`, `PRO_FIAT_FILE`, `PRO_CATIA_DIRECT_FILE`, or `PRO_ACIS_FILE`, set the following argument:
  - *arg2*—The integer pointer to an odd or even number.
- For `PRO_CGM_FILE`, set the following arguments:
  - *arg2*—Represents the integer pointer to the export type `PRO_EXPORT_CGM_CLEAR_TEXT` or `PRO_EXPORT_CGM_MIL_SPEC`.
  - *arg3*—Represents the integer pointer to the scalar type `PRO_EXPORT_CGM_ABSTRACT` or `PRO_EXPORT_CGM_METRIC`.
- For `PRO_CONNECTOR_PARAMS`, set the following arguments:

- *arg1*—Represents the integer pointer to `ProIdTable`. `ProIdTable` is an integer array of component identifiers.
- *arg2*—Represents the integer pointer to the number of component identifiers.

- For `PRO_CABLE_PARAMS_FILE`, set the following arguments:
  - *arg1*—Represents a `ProSolid` (part pointer).
  - *arg2*—Represents the cable name.
- For `PRO_DIFF_REPORT_FILE`, set the following argument:
  - *arg4*—Represents the model to which the input model is compared to generate the difference report.
- For `PRO_RELATION_FILE`, set the following argument:
  - *arg2*—Represents the individual feature relations. It is an integer pointer to the feature identifier that gets the relations contained in a feature. If this is `NULL` you get the relations contained in the model.

Exporting 2D Models

Functions Introduced:

- **Pro2dExport()**
- **Pro2dExportdataAlloc()**
- **Pro2dExportdataFree()**
- **Pro2dExportdataSheetoptionSet()**
- **Pro2dExportdataSheetsSet()**
- **Pro2dExportdataModelspacesheetSet()**
- **ProProductviewexportoptsAlloc()**
- **ProProductviewexportoptsFree()**
- **ProProductviewexportoptsFormatSet()**
- **ProProductviewFormattedMdlnameExport()**
- **ProPrintPrinterOptionsGet()**
- **ProPrintMdlOptionsGet()**
- **ProPrintPlacementOptionsGet()**
- **ProPrintPCFOptionsGet()**
- **ProPrintExecute()**

Export Format	Creo Parametric TOOLKIT Functions	Type Constant
STEP	Pro2dExport()	PRO_STEP_FILE
IGES		PRO_IGES_FILE
MEDUSA		PRO_MEDUSA_FILE
DXF		PRO_DXF_FILE
DWG		PRO_DWG_FILE
CGM		PRO_CGM_FILE
TIFF		PRO_SNAP_TIFF_FILE
Stheno		PRO_STHENO_FILE
DXF	ProOutputFileMdlnameWrite()	PRO_DXF_FILE
DWG		PRO_DWG_FILE
CGM file		PRO_CGM_FILE
PVS file, Plot file	ProProductviewFormatted Export()	PRO_PV_FORMAT_PVS
ED file, Plot file		PRO_PV_FORMAT_ED
EDZ file		PRO_PV_FORMAT_EDZ
PVZ file		PRO_PV_FORMAT_PVZ
Plot file	ProPrintExecute()	N/A

The function Pro2dExport() exports existing two-dimensional models into a single object file. The exported model can be a single drawing, notebook or diagram, or multiple sheets of a drawing. It supports the STEP, SET, IGES, Medusa, DXF, CGM, TIFF, Stheno,and DWG formats. The interface file obtained using the function Pro2dExport() is controlled by one of its input argument *data,* an instance of the Pro2dExportdata object. Note that the *data* argument is optional; you do not have to specify it when exporting only the current sheet of the 2D model. Additionally, several Creo Parametric configuration options related to entity type export options can affect the results of the export operation. Refer to the Creo Parametric Online Help for details on the configuration options.

Example 1 Publishing a Drawing

The sample code in UgDwgPublishContext.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dwg shows how to publish a drawing in the given context.

The functions Pro2dExportdataAlloc() and Pro2dExportdataFree() allocate and free the memory for the Pro2dExportdata object containing the 2D export options and flags.

The function Pro2dExportdataSheetoptionSet() assigns the sheet export option for export to the specified 2D format. The sheet export option can take one of the following values:

- PRO2DEXPORT_CURRENT_TO_MODEL_SPACE—Specifies that only the drawing's current sheet will be exported as model space into the specified 2D format. This is the default value.
- PRO2DEXPORT_CURRENT_TO_PAPER_SPACE—Specifies that only the drawing's current sheet will be exported as paper space into the specified 2D format. This value is only available for formats that support the concept of model space and paper space, for example, DXF and DWG.
- PRO2DEXPORT_ALL—Specifies that all the sheets in a drawing will be exported into the specified 2D format as paper space, if applicable to the format type.
- PRO2DEXPORT_SELECTED—Specifies that selected sheets in a drawing will be exported as paper space and one sheet will be exported as model space.

The function Pro2dExportdataSheetsSet() assigns the sheet numbers to be exported as paper space to the specified 2D export format file. Use this function only if the sheet export option is set to PRO2DEXPORT_SELECTED.

The function Pro2dExportdataModelspacesheetSet() assigns the sheet number to be exported as model space. Use this function only if the export format supports the concept of model space and paper space, and if the sheet export option is set to PRO2DEXPORT_SELECTED.

The functions ProProductviewexportoptsAlloc() and ProProductviewexportoptsFree() allocate and free the memory assigned to the ProProductviewExportOptions object containing the Creo View export formats.

The function ProProductviewexportoptsFormatSet() assigns the flag specifying the Creo View export format.

The function ProProductviewFormattedMdlnameExport() exports a drawing to one of the following user-defined Creo View formats:

- PRO_PV_FORMAT_PVS
- PRO_PV_FORMAT_ED
- PRO_PV_FORMAT_EDZ
- PRO_PV_FORMAT_PVZ

Use the function ProPrintPrinterOptionsGet() to get the options for a specified printer. Specify the printer type as the input argument for this function. The supported printer types are:

- POSTSCRIPT—Generic Postscript
- COLORPOSTSC—Color Postscript
- MS_PRINT_MGR—MS Print Manager

Note

For a list of all supported printers, please refer to the **Add Printer Type** list in the **Printer Configuration** dialog box of Creo Parametric.

The function gets the initialized printer options. The options include the file related options, print command options and printer specific options as follows:

- File Related:
  - save_to_file—Saves a plot to a file.
  - save_method—Specifies if the plot is to be appended to a file, saved to a single file, or saved to multiple files.
  - filename—Specifies the name of the file to which the plot is saved.
  - delete_after—Deletes the plot file after printing.
- Print Command:
  - send_to_printer—Sends the plot directly to the printer.
  - print_command—Specifies the complete command that you want to use for printing.
  - pen_table—Specifies the complete path to the file containing the pen table.
  - paper_size—Specifies the size of the paper to be printed.
  - quantity—Specifies the number of copies to be printed.
- Printer Specific:
  - sw_handshake—Specifies the type of the handshake initialization sequence. Specify the value as `True` to set the initialization sequence to Software and as `False` to set it to Hardware.

Note

Consult your system administrator for more information on handshaking.

  - roll_media—Specifies whether to use roll-media or cut-sheet.
  - use_ttf—Specifies whether to use TrueType font or stroked text.
  - slew—Specifies the speed of the pen in centimeters per second in X and Y direction.
  - rotate_plot—Specifies that the plot is to be rotated by 90 degrees.

Use the function `ProPrintMdlOptionsGet()` to get the initialized model options for the model to be printed. The available model options are:

- mdl—Specifies the model to be printed.
- quality—Determines the quality of the model to be printed. It checks for no line, no overlap, simple overlap, and complex overlap.

The model options specific to drawing objects are:

- use_drawing_size—Overrides the size of the paper specified in the Printer options.
- draw_format—Prints the drawing format used.
- segmented—If true, that is the value is set to a boolean of 1, the printer prints drawing full size, but in segments that are compatible with the selected paper size. This option is available only if you are plotting a single page.
- layer_only—Prints the specified layer only.
- layer_name—Prints the name of the layer.
- sheets—Prints the current sheet, all sheets, or selected sheets.
- range—An array of two integers specifying the start and end sheet numbers.

The model option specific to solid objects is:

- use_solid_scale—Prints using the scale used in the solid model.

Use the function `ProPrintPlacementOptionsGet()` to get the current print placement options such as print scale, offset, zoom, and so on. The options available for the object placement are:

- Placement Options:
  - scale—Specifies the scale used for the selected plot.
  - offset—An array of two doubles representing the offset from the lower-left corner of the plot.
  - keep_panzoom—Maintains the pan and zoom values of a window.
- Clipping Options:
  - clip_plot—Specifies whether you want to clip the plot.
  - shift_to_ll_corner—Shifts the clip area to the lower-left corner of the plot
  - clip_area—Two dimensional array of four double representing the area that is clipped. The range of the values of this option is 0.0 through 1.0.
- Label Options:
  - place_label—Specifies whether you want to place the label on the plot.
  - label_height—Height of the label in inches.

Use the function `ProPrintPCFOptionsGet()` to get the print options from a specified Plotter Configuration File. Specify the name of the plotter configuration file and the name of the model to be printed. The function gets the printer options, model options and placement options.

Use the function `ProPrintExecute()` to print a Creo Parametric window using the specified printer options, model options and placement options. The drawing must be displayed in a window to be successfully printed.

Automatic Printing of 3D Models

Creo Parametric TOOLKIT provides the capability of automatically creating and plotting a drawing of a solid model. The Creo Parametric TOOLKIT application needs only to supply instructions for the print activity, and Creo Parametric will automatically create the drawing, print it, and then discard it.

The methods listed here are analogous to the command **File ► Print ► Quick Drawing** in Creo Parametric's user interface.

Functions Introduced:

- **ProQuickprintoptionsAlloc()**
- **ProQuickprintoptionsFree()**
- **ProQuickprintoptionsLayouttypeSet()**
- **ProQuickprintoptionsOrientationSet()**
- **ProQuickprintoptionsSizeSet()**
- **ProQuickprintoptionsViewAdd()**
- **ProQuickprintoptionsThreeviewlayoutSet()**
- **ProQuickprintoptionsProjectionsSet()**
- **ProQuickprintoptionsTemplateSet()**
- **ProQuickprintoptionsPrintFlatToScreenAnnotsSet()**
- **ProQuickprintExecute()**

The function `ProQuickprintoptionsAlloc()` allocates a quick drawing options handle.

Use the function `ProQuickprintoptionsFree()` to free a quick drawing options handle.

Use the function `ProQuickprintoptionsLayouttypeSet()` to assign the layout type for the quick drawing operation. You can either specify a drawing layout using the instructions or use a template to define the drawing. The following are the available layout types:

- PRO_QPRINT_LAYOUT_PROJ—Use a projected view-type layout.
- PRO_QPRINT_LAYOUT_MANUAL—Use a manually arranged layout.
- PRO_QPRINT_LAYOUT_TEMPLATE—Use a drawing template to define the layout. If this option is used, only the template name is needed to define the print; other options are not used.

Use the function ProQuickprintoptionsOrientationSet() to assign the sheet orientation for the quick drawing operation. The following are the available sheet orientation types:

- PRODEV_ORIENTATION_PORTRAIT
- PRODEV_ORIENTATION_LANDSCAPE

Use the function ProQuickprintoptionsSizeSet() to assign the size of the print for the quick drawing operation. ProPlotPaperSize specifies the paper size and can be any of the following types:

- A_SIZE_PLOT
- B_SIZE_PLOT
- C_SIZE_PLOT
- D_SIZE_PLOT
- E_SIZE_PLOT
- A4_SIZE_PLOT
- A3_SIZE_PLOT
- A2_SIZE_PLOT
- A1_SIZE_PLOT
- A0_SIZE_PLOT
- F_SIZE_PLOT

Note

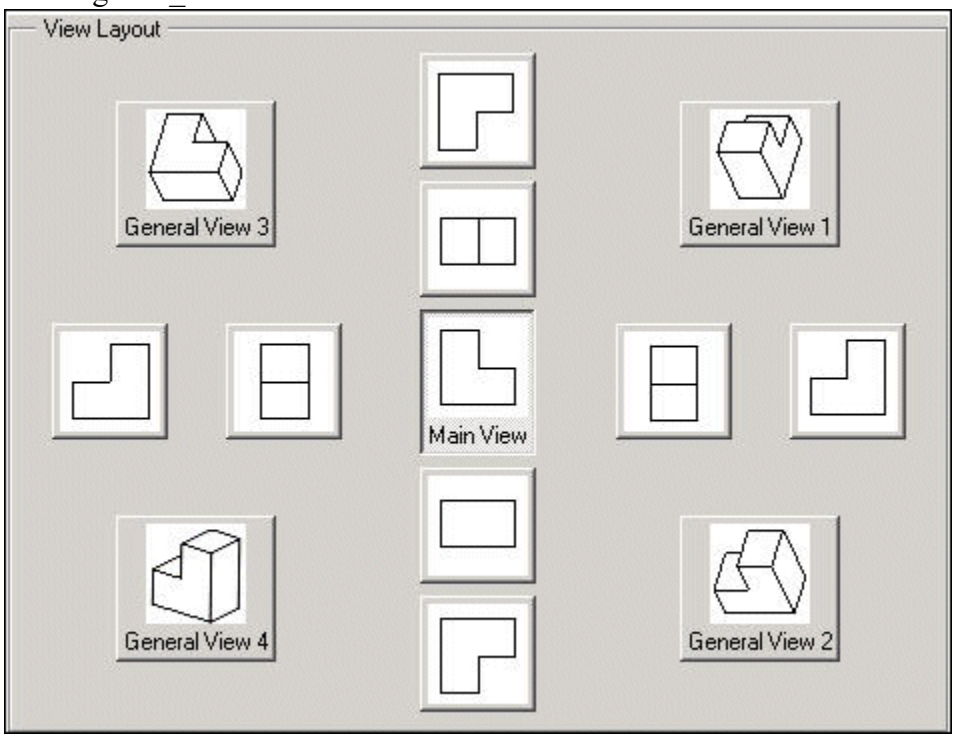
Variable size plots are not supported by this utility.

Use the function ProQuickprintoptionsViewAdd() to add a new general view. The input arguments of this function are:

- *options*—Specifies the options handle.
- *location*—Specifies the location of the view being added for projected view layout. This option is ignored for a manual view layout. It can be of the following types:
  - PRO_QPRINTPROJ_GENVIEW_MAIN

- PRO_QPRINTPROJ_GENVIEW_NW
- PRO_QPRINTPROJ_GENVIEW_SW
- PRO_QPRINTPROJ_GENVIEW_SE
- PRO_QPRINTPROJ_GENVIEW_NE

The general view location options are analogous to the locations in the quick drawing user_interface:



Note

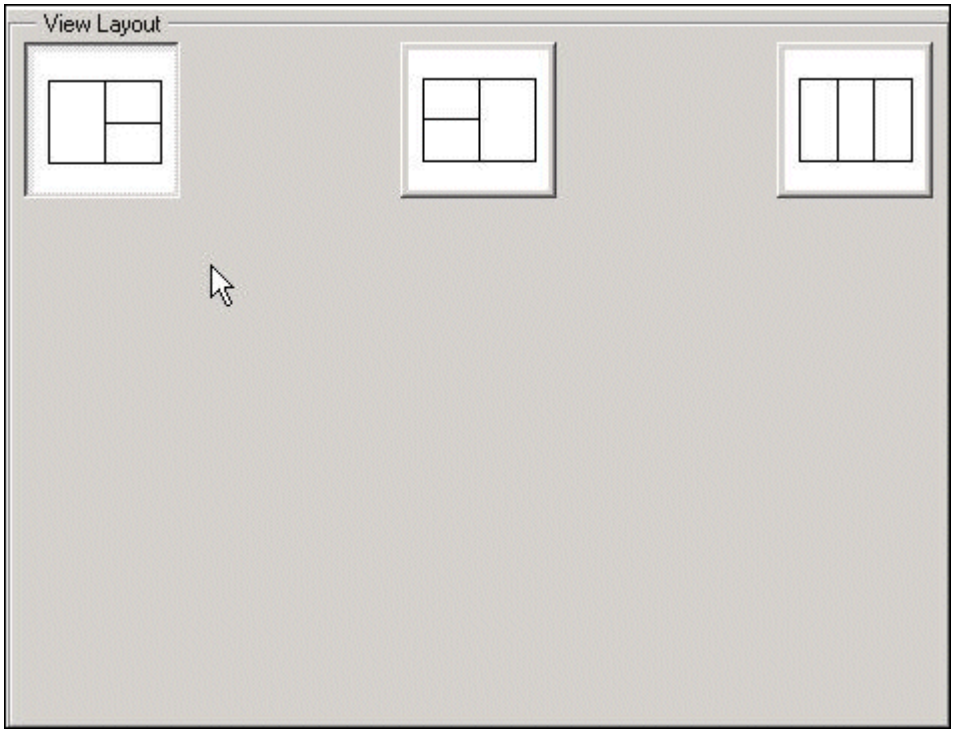
For manual view layouts, the order in which the views are added determine their final location in the drawing. For this configuration, PTC does not support using more than four views.

- *view_name*—Specifies the name of the saved model view.
- *scale*—Specifies the view scale.
- *display_style*—Specifies the view display style to use and is of the following types:
  - `PRO_DISPSTYLE_DEFAULT`
  - `PRO_DISPSTYLE_WIREFRAME`
  - `PRO_DISPSTYLE_HIDDEN_LINE`
  - `PRO_DISPSTYLE_NO_HIDDEN`
  - `PRO_DISPSTYLE_SHADED`
  - `PRO_DISPSTYLE_FOLLOW_ENVIRONMENT`
  - `PRO_DISPSTYLE_SHADED_WITH_EDGES`

Use the function `ProQuickprintoptionsThreeviewlayoutSet()` to assign the layout type when three views are being used in a manual layout (`PRO_QPRINT_LAYOUT_MANUAL`). The layout can be either of the following types:

- `PRO_QPRINTMANUAL_3VIEW_1_23VERT`
- `PRO_QPRINTMANUAL_3VIEW_23_VERT1`
- `PRO_QPRINTMANUAL_3VIEW_123_HORIZ`

These options correspond to the diagrams in the user interface:



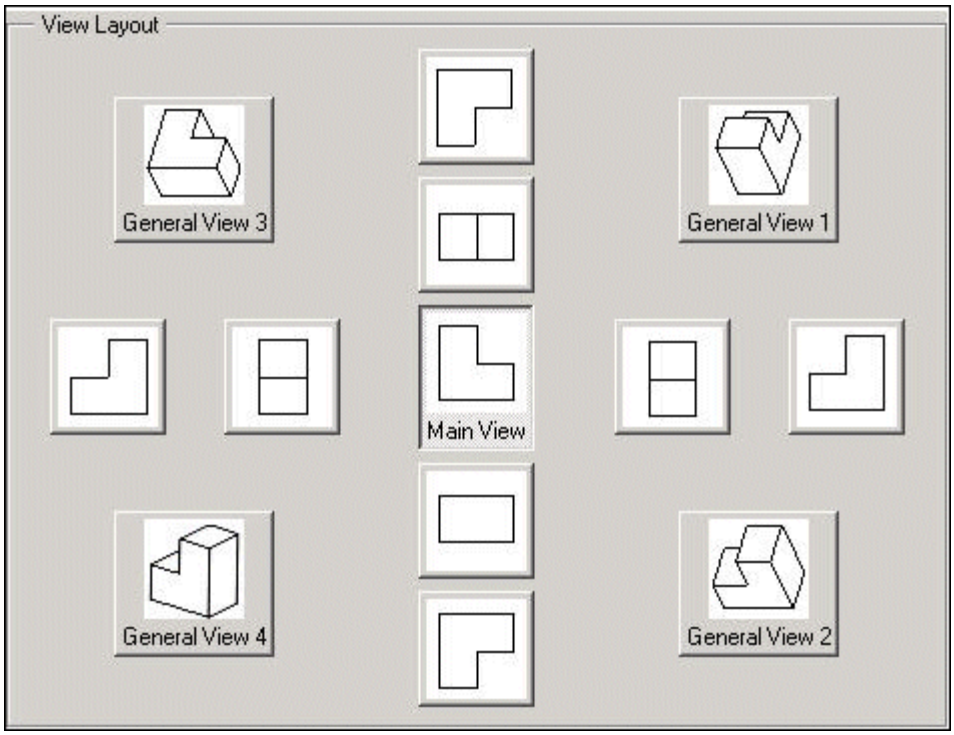
Use the function `ProQuickprintoptionsProjectionsSet()` assign the projected views to be included in the quick drawing operation. The function applies only to `PRO_QPRINT_LAYOUT_PROJ`. The projections are of the following types:

- `PRO_QPRINTPROJ_TOP_VIEW`
- `PRO_QPRINTPROJ_RIGHT_VIEW`
- `PRO_QPRINTPROJ_LEFT_VIEW`
- `PRO_QPRINTPROJ_BOTTOM_VIEW`
- `PRO_QPRINTPROJ_BACK_NORTH`
- `PRO_QPRINTPROJ_BACK_EAST`
- `PRO_QPRINTPROJ_BACK_SOUTH`
- `PRO_QPRINTPROJ_BACK_WEST`

Note

Projection views takes the same view scale and display style as the main view.

The options correspond to the projected members of the diagram in the user interface:



Use the function `ProQuickprintoptionsTemplateSet()` to assign the path to the drawing template file to be used for the quick drawing operation. The function applies only to layout type `PRO_QPRINT_LAYOUT_TEMPLATE`.

Note

The quick drawing operation shows the exact image of the model as is shown on-screen. Therefore, if the drawing template has drawing views set with display options such as view clipping, simplified representations, or layers, these settings are ignored while plotting. The resulting plot reflects whatever is seen on-screen.

Use the function `ProQuickprintoptionsPrintFlatToScreenAnnotsSet()` to set the `ProBoolean` flag to print flat-to-screen annotations. The flat-to-screen annotations created at screen locations in the Creo Parametric graphics window are printed at their relative locations in the drawing. You can print flat-to-screen annotations such as notes, symbols, and surface finish symbols.

Once the instructions have been prepared, use the function `ProQuickprintExecute()` to execute a print operation for a given solid model. It has the following input arguments:

- *solid*—Specifies the solid model to be printed.
- *pcf_path*—Specifies the path to the plotter configuration file to use. If no path is specified, then the path will have the value of the configuration option `quick_print_plotter_config_file`.
- *options*—Specifies the details of the quick drawing operation given by the `ProQuickprintOptions` handle.

Exporting 3D Models

Creo Parametric TOOLKIT provides export capabilities for three dimensional geometry to various formats.

Functions Introduced:

- **ProIntf3DFileWrite()**
- **ProIntf3DFileWriteWithDefaultProfile()**
- **ProIntf3DLayerSetupFileSet()**
- **ProIntf3DLayerSetupFileIsIgnored()**
- **ProIntf3DCsysSet()**
- **ProIntf3DCsysIsIgnored()**
- **ProIntf3DModelDataClear()**
- **ProOutputBrepRepresentationAlloc()**
- **ProOutputBrepRepresentationFlagsSet()**
- **ProIntfExportProfileLoad()**
- **ProOutputBrepRepresentationIsSupported()**
- **ProOutputBrepRepresentationFree()**
- **ProOutputInclusionAlloc()**
- **ProOutputInclusionFacetparamsSet()**
- **ProOutputInclusionWithOptionsSet()**
- **ProOutputInclusionFlagsSet()**
- **ProOutputInclusionFree()**
- **ProOutputLayerOptionsAlloc()**
- **ProOutputLayerOptionsAutoidSet()**
- **ProOutputLayerOptionsSetupfileSet()**
- **ProOutputLayerOptionsFree()**
- **ProOutputAssemblyConfigurationIsSupported()**
- **ProRasterFileWrite()**

- **ProIntfSliceFileWithOptionsMdlnameExport()**
- **ProExportVRML()**
- **ProProductviewexportoptsAlloc()**
- **ProProductviewexportoptsFree()**
- **ProProductviewexportoptsFormatSet()**
- **ProProductviewFormattedMdlnameExport()**

Export Format	Creo Parametric TOOLKIT Functions	Type Constant
STEP file (Standard for the Exchange of Product Model Data)	`ProIntf3DFileWrite()` `ProIntf3DFileWriteWithDefaultProfile()`	`PRO_INTF_EXPORT_STEP`
VDA file		`PRO_INTF_EXPORT_VDA`
IGES (3D) file		`PRO_INTF_EXPORT_IGES`
CATIA (.model) file		`PRO_INTF_EXPORT_CATIA_MODEL`
SAT file (ACIS format for Creo Parametric)		`PRO_INTF_EXPORT_SAT`
NEUTRAL file (ASCII text)		`PRO_INTF_EXPORT_NEUTRAL`
CADDS file		`PRO_INTF_EXPORT_CADDS`
CATIA (.session) file		`PRO_INTF_EXPORT_CATIA_SESSION`
Parasolid file		`PRO_INTF_EXPORT_PARASOLID`
UG file		`PRO_INTF_EXPORT_UG`
CATIA V5 Part file		`PRO_INTF_EXPORT_CATIA_PART`
CATIA V5 Assembly file		`PRO_INTF_EXPORT_CATIA_PRODUCT`
JT Open format		`PRO_INTF_EXPORT_JT`
CATIA Graphical Representation (CGR) format		`PRO_INTF_EXPORT_CATIA_CGR`
DWG file		`PRO_INTF_EXPORT_DWG`
DXF file		`PRO_INTF_EXPORT_DXF`
SolidWorks Part File		`PRO_INTF_EXPORT_SW_PART`
SolidWorks Assembly File		`PRO_INTF_EXPORT_SW_ASSEM`
3D Manufacturing Format (3MF)		`PRO_INTF_EXPORT_3MF`
CATIA facets file	`ProIntfSliceFileWithOptionsMdlnameExport()`	`PRO_CATIAFACETS_FILE`
INVENTOR file		`PRO_INVENTOR_FILE`
Render file		`PRO_RENDER_FILE`
SLA ASCII file		`PRO_SLA_ASCII_FILE`
SLA Binary file		`PRO_SLA_BINARY_FILE`
Additive manufacturing file		`PRO_AMF_FILE`
JPEG file	`ProRasterFileWrite()`	`PRORASTERTYPE_JPEG`
BMP file		`PRORASTERTYPE_BMP`
TIFF file		`PRORASTERTYPE_TIFF`

Export Format	Creo Parametric TOOLKIT Functions	Type Constant
EPS file (Postscript)		PRORASTERTYPE_EPS
PVS file, OL file (a separate OL file is created for each PART in an assembly)	ProProductviewFormattedExport()	PRO_PV_FORMAT_PVS
ED file, OL file (a separate OL file is created for each PART in an assembly)		PRO_PV_FORMAT_ED
EDZ file		PRO_PV_FORMAT_EDZ
PVZ file		PRO_PV_FORMAT_PVZ
VRML	ProExportVRML()	N/A
Shrinkwrap	ProSolidShrinkwrapCreate()	N/A

The following data is included during the export of Creo Parametric models to other formats:

- 3D Manufacturing Format (3MF)—From Creo Parametric 5.0.1.0 onward, you can export Creo Parametric models to the 3MF format. The export includes part-level colors, top-assembly parameters, facet geometry, and hidden entities.
- JT—Creo Parametric models are exported to JT with their color overrides. Components with color overrides at any level in an assembly structure are supported.

  From Creo Parametric 3.0 onward, the Product Manufacturing Information (PMI) of the annotations is exported as semantic representation from Creo Parametric to JT models. The semantic export is supported only for 3D notes and basic dimensions. All the other types of annotations are exported as graphical entities. You can export planar and zonal cross-sections attached to combined states from Creo Parametric files to JT.

  Note

  From Creo Parametric 2.0 M200 onward the license INTF_for_JT is required to export a Creo Parametric model to JT. If the license is not available the functions return the error PRO_TK_NO_LICENSE.

- Creo View—You can export colors assigned to the components of assemblies and their sub-assembly models, including the colors of the sub-level entities such as parts, quilts, and faces from Creo Parametric to Creo View.

Creo Parametric models are exported to Creo View with their color overrides. Components with color overrides at any level in an assembly structure are supported. Along with components, color overrides are also supported for component model items, such as, face and quilts.

- SolidWorks—From Creo Parametric 3.0 onward, you can export Creo Parametric models to SolidWorks. The export includes basic geometry such as datum features, colors, attributes, and layers, part models, assembly structures, boundary representation geometry, and non-geometric data.
- Unigraphics—The export of Creo Parametric models to Unigraphics includes the export of basic geometry such as datum features, colors, attributes, and layers.

Note

Refer to the Creo Parametric Data Exchange Help for more information on exporting geometry from Creo Parametric to other formats. Refer to the compatibility matrix on PTC.com for the supported software versions.

The following functions will be deprecated in a future release of Creo Parametric. Use the function `ProIntf3DFileWriteWithDefaultProfile()` instead to export Creo Parametric models to other file formats. All the options that can be set with these functions, can also be set using the export profile option in Creo Parametric. Refer to the Creo Parametric Data Exchange Online Help for more information.

- `ProIntf3DFileWrite()`
- `ProOutputInclusionAlloc()`
- `ProOutputInclusionFree()`
- `ProOutputInclusionFlagsSet()`
- `ProOutputLayerOptionsAlloc()`
- `ProOutputLayerOptionsAutoidSet()`
- `ProOutputLayerOptionsSetupfileSet()`
- `ProOutputLayerOptionsFree()`
- `ProOutputInclusionWithOptionsSet()`

The function `ProIntf3DFileWriteWithDefaultProfile()` exports a Creo Parametric model to the specified output format using the default export profile.

The function `ProIntf3DFileWrite()` will be deprecated in a future release of Creo Parametric. Use the function ProIntf3DFileWriteWithDefaultProfile() instead. The function `ProIntf3DFileWrite()` exports a Creo Parametric model to the specified output format. The following types of output formats are supported:

- STEP
- VDA
- IGES
- CATIA MODEL
- SAT (ACIS format in Creo Parametric)
- NEUTRAL
- CADDS
- CATIA SESSION
- PARASOLID
- UG
- CATIA V5
- JT Open
- CATIA Graphical Representation
- DWG
- DXF

While exporting the model, you can specify the structure and contents of the output files as:

- Flat File—Exports all of the geometry of the assembly to a single file as if it were a part. This is similar to the Single File format in Creo Parametric for STEP output.
- Single File—Exports an assembly structure to a file with external references to component files. This file contains only top-level geometry. This is similar to the Dittos format in Creo Parametric for CATIA, Separate Parts Only for STEP and One Level for IGES outputs. A part or an assembly is exported as a single file for the DXF and the DWG formats.
- Multi Files—Exports an assembly structure to a single file and the components to component files. It creates component parts and subassemblies with their respective geometry and external references. This option supports

all levels of hierarchy. This is similar to All Levels format for IGES and Separate All Parts for STEP in Creo Parametric.

- Parts—Exports an assembly as multiple files containing geometry information of its components and assembly features. This is similar to All Parts format for IGES in Creo Parametric.

Some output formats support only certain types of assembly configurations. The default assembly configuration is a flat file.

Note

Using the Creo Parametric TOOLKIT function `ProIntf3DFileWrite()` you can export a Creo Parametric 3D model to a JT file format.

From Creo Parametric 2.0 M200 onward, the function `ProIntf3DFileWrite()` exports a Creo Parametric 3D model to JT file format only if the license `INTF_for_JT` is available. If the license is not available the function returns the error `PRO_TK_NO_LICENSE`.

Starting with Creo Parametric1.0, XT-brep data format is supported along with the JT-brep format for storing the 3D model data to export. Use the new configuration option `intf3d_ out_jt_brep` to export the Creo Parametric model using the function ProIntf3DFileWrite(). This configuration option takes the following values:

- `NO`—This is the default value. When you set the configuration option `intf3d_out_jt_brep` as NO, the function `ProIntf3DFileWrite()` exports the Creo Parametric model to JT format as facet representation only.
- `JT_BREP`—When you set the configuration option `intf3d_out_jt_brep` to `JT_BREP`, the exported Creo Parametric model has both the JT-brep format and the faceted representation.
- `XT_BREP`—When you set the configuration option `intf3d_out_jt_brep` to `XT_BREP`, the exported Creo Parametric model has both the XT-brep format and the faceted representation.

- The export of a Creo Parametric 3D model to a JT file format using the function `ProIntf3DFileWrite()` is also impacted by the following configuration options:
  - `intf3d_out_jt_auto_lods`—It takes the values `yes` or `no*`. If you set the configuration option `intf3d_out_jt_auto_lods` to `yes`, you can export up to three Levels of Detail (LODs) to the JT format.
  - `intf3d_out_jt_config_name`—Name of the JT configuration file. You can define a configuration file `jt.config`. You can define parameters and export options in this file. You can also control the export of the LODs to the JT format via Creo Parametric TOOLKIT using this file. The options set in this file override the setting of the configuration option `intf3d_out_jt_auto_lods`. Refer to the section Export Options in the JT Configuration File on page 691 for more information about the options that can be set in the `jt.config` file.
  - `intf3d_out_jt_config_path`—Path of the JT configuration file.

- The configuration option `intf3d_out_export_as_facets` is now obsolete.

The function `ProIntf3DLayerSetupFileSet()` sets the layer setup file for the export. The input arguments follow:

- *model*—The model used for export.
- *layer_setup_file*—The full path of the input layer setup file. Pass the value as `NULL` to set default layer setup settings for input `file_type`. Layer setup file is not supported for `PRO_INTF_EXPORT_CADDS` and `PRO_INTF_EXPORT_NEUTRAL` file types.

The function `ProIntf3DLayerSetupFileIsIgnored()` checks if layer setup file is ignored or not during export. This function returns if the layer setup file is used for the last export, using the function `ProIntf3DFileWriteWithDefaultProfile()`.

For reliable results, call the function `ProIntf3DFileWriteWithDefaultProfile()` before calling `ProIntf3DLayerSetupFileIsIgnored()`.

The output argument `is_ignored` returns `PRO_B_TRUE` if the layer setup file is ignored and returns `PRO_B_FALSE` if layer setup file is not ignored.

The function `ProIntf3DCsysSet()` sets the reference coordinate system Csys for the export. The input arguments follow:

- *model*—The model used for export.
- *csys_sel*—The reference coordinate system. Pass the value as `NULL` to set default coordinate system. Reference Csys is not supported for `PRO_INTF_EXPORT_CADDS` and `PRO_INTF_EXPORT_NEUTRAL` file types.

The function `ProIntf3DCsysIsIgnored()` checks if the reference coordinate system is ignored or not during export. The function returns if the reference Csys is used for the last export, using the function `ProIntf3DFileWriteWithDefaultProfile()`.

For reliable results, call the function `ProIntf3DFileWriteWithDefaultProfile()` before calling `ProIntf3DLayerSetupFileIsIgnored()`.

The output argument `is_ignored` returns `PRO_B_TRUE` if the reference Csys is ignored and returns `PRO_B_FALSE` if the reference Csys is not ignored.

The function `ProIntf3DModelDataClear()` clears the model data set by the functions `ProIntf3DLayerSetupFileSet()` and `ProIntf3DCsysSet()`.

Use the function `ProOutputAssemblyConfigurationIsSupported()` to check if the specified assembly configuration is valid for the particular model and the specified export format. This function must be called before exporting the model to the specified output format using the function `ProIntf3DFileWrite()` except for the CADDS and STEP2D formats.

The function `ProOutputBrepRepresentationAlloc()` allocates memory for the geometric representation data structure. This data structure represents the types of geometry supported by the export operation. The types of geometric representations are:

- Wireframe
- Surfaces
- Solid
- Quilts (Shell in Creo Parametric)

These correspond to the options shown in the Creo Parametric dialog box for export. Note that some formats allow a combination of types to be input.

The function `ProOutputBrepRepresentationFlagsSet()` sets the flags for the geometric representation data structure. It specifies the type of geometry to be exported.

The function `ProIntfExportProfileLoad()` loads the specified profile for export. You can use this function when you want to use the export profile of your choice instead of the default export profile in a particular Creo Parametric session. The input argument *profile* is the full path to the profile along with the profile name and extension.

Note

Once the export profile file is loaded in a Creo Parametric session, it will be active in the interactive mode as well.

The function `ProOutputBrepRepresentationIsSupported()` checks if the specified geometric representation is valid for a particular export format. This function should be called before exporting the model to the specified output format using the function `ProIntf3DFileWrite()`, to check if the planned configuration is supported by the Creo Parametric interface options.

The function `ProOutputBrepRepresentationFree()` frees the memory allocated for the geometry data structure.

The function `ProOutputInclusionAlloc()` will be deprecated in a future release of Creo Parametric. The function `ProOutputInclusionAlloc()` allocates memory for the inclusion structure to be used while exporting the model.

The function `ProOutputInclusionFlagsSet()` will be deprecated in a future release of Creo Parametric. It is recommended that you set this option in export profile file in Creo Parametric. The function `ProOutputInclusionFlagsSet()` determines whether to include certain entities during export. The types of entities are:

- Datums—Determines whether datum curves are included when exporting files. If the flag is set to true the datum curve and point information is included during export. The default value is false.
- Blanked—Determines whether entities on blanked layers are exported. If the flag is set to true, entities on blanked layers are exported. The default value is false.
- Facets—Determines whether faceted geometry is included when exporting the models. The default value of the flag is false.

The function `ProOutputInclusionFacetparamsSet()` assigns the parameters to use while exporting the model to a faceted format such as `PRO_INTF_EXPORT_CATIA_CGR`. These parameters are as follows:

- *chord_height*—The chord height to use for the exported facets.
- *angle_control*—The angle control to use for the exported facets.

Note

The function `ProOutputInclusionFacetparamsSet()` has been deprecated. Use the function `ProOutputInclusionWithOptionsSet()` instead.

The function `ProOutputInclusionWithOptionsSet()` will be deprecated in a future release of Creo Parametric. It is recommended that you set this option in export profile file in Creo Parametric. Use the function `ProOutputInclusionWithOptionsSet()` to set the parameters and configuration flags used while exporting the model to a faceted format such as `PRO_INTF_EXPORT_CATIA_CGR`. The input arguments are as follows:

- *parameters*—Specifies a `ProArray` of parameters that consists of the following three elements:
  - *chord_height*—The chord height of the exported facets.
  - *angle_control*—The angle control of the exported facets. Specify a value between 0.0 to 1.0. If the angle control is out of bounds, Creo Parametric changes it to the closest limit without returning an error.
  - *step_size*—The step size of the exported facets. If the step size is less or equal to 0, it is ignored.

Note

If the chord height or step size are too small or too big, then Creo Parametric resets it to the smallest or biggest acceptable value, respectively, without returning an error.

- *config_flags*—Specifies the configuration flags that control the export operation. They are as follows:
  - `PRO_FACET_STEP_SIZE_OFF`—Switches off the step size control.
  - `PRO_FACET_FORCE_INTO_RANGE`—Forces the out-of-range parameters into range. If any of the `PRO_FACET_*_DEFAULT` option is set, then the option `PRO_FACET_FORCE_INTO_RANGE` is not applied on that parameter.
  - `PRO_FACET_STEP_SIZE_ADJUST`—Adjusts the step size according to the component size.
  - `PRO_FACET_CHORD_HEIGHT_ADJUST`—Adjusts the chord height according to the component size.
  - `PRO_FACET_USE_CONFIG`—If this flag is set, values of the flags `PRO_FACET_STEP_SIZE_OFF`, `PRO_FACET_STEP_SIZE_ADJUST`, and `PRO_FACET_CHORD_HEIGHT_ADJUST` are ignored and the configuration settings from the Creo Parametric user interface are used during the export operation
  - `PRO_FACET_CHORD_HEIGHT_DEFAULT`—Uses the default value set in the Creo Parametric user interface for the chord height.
  - `PRO_FACET_ANGLE_CONTROL_DEFAULT`—Uses the default value set in the Creo Parametric user interface for the angle control.
  - `PRO_FACET_STEP_SIZE_DEFAULT`—Uses the default value set in the Creo Parametric user interface for the step size.
  - `PRO_FACET_INCLUDE_QUILTS`—Includes quilts in the export of Creo Parametric model to the specified format.

- PRO_EXPORT_INCLUDE_ANNOTATIONS—Includes annotations in the export of Creo Parametric model to the specified format.

  Note

  To include annotations, during the export of Creo Parametric model, you must call the function `ProMdlDisplay()` before calling `ProIntf3DFileWrite()`.

- PRO_FACET_VISIBLE_MODELS—Exports models which have their visibility status set to **Show** in Creo Parametric. Models that are hidden are not exported.

Note

The behavior of the function `ProOutputInclusionWithOptionsSet()` is similar to the function `ProOutputInclusionFacetparamsSet()` if the configuration flag `PRO_FACET_STEP_SIZE_OFF` is set.

The function `ProOutputInclusionFree()` will be deprecated in a future release of Creo Parametric. The function `ProOutputInclusionFree()` frees the memory allocated for the inclusion structure.

The function `ProOutputLayerOptionsAlloc()` will be deprecated in a future release of Creo Parametric. The function `ProOutputLayerOptionsAlloc()` allocates memory for the layer options data structure. The layer options are:

- AutoId—A flag indicating whether layers should be automatically assigned numerical ids when exporting.
- LayerSetupFile—The layer setup file contains the name of the layer, its display status, the interface ID and number of sub layers.

Specify the name and complete path of the layer setup file. This file contains the layer assignment information.

The function `ProOutputLayerOptionsAutoidSet()` will be deprecated in a future release of Creo Parametric. It is recommended that you set this option in export profile file in Creo Parametric. The function `ProOutputLayerOptionsAutoidSet()` enables you to set or remove an interface layer ID. If true, automatically assigns interface ids to layers not assigned ids and exports them. The default value is false.

The function ProOutputLayerOptionsSetupfileSet() will be deprecated in a future release of Creo Parametric. It is recommended that you set this option in export profile file in Creo Parametric. Use the function ProOutputLayerOptionsSetupfileSet() to specify the name and complete path of the layer setup file.

The function ProOutputLayerOptionsFree() will be deprecated in a future release of Creo Parametric. The function ProOutputLayerOptionsFree() frees the memory allocated for the layer options structure.

Use function ProRasterFileWrite() to create a standard Creo Parametric raster output file. Note that this function does not support output of drawings (2-dimensional objects) in Drawing mode.

The PRO_SPOOL_FILE option reads in a Diagram spool file.

The function ProIntfSliceFileWithOptionsMdlnameExport() exports to tesellated formats such as STL, Render, AMF, Inventor, CatiaFacets, 3MF, and Optegra Visualizer based on the values of a ProArray of parameters and two configuration flags. These formats require the maximum chord height, angle control, and transformation to be specified for the model being exported. If the specified model is an assembly, the last input argument of the function is the component path; if the model is a part, this argument is NULL. These parameters and configuration flags are same as the ones assigned by the function ProOutputInclusionWithOptionsSet() described earlier in this section. Refer to its description for more information on the parameters and configuration flags.

The function ProExportVRML() exports a solid from a Creo Parametric session, or a Creo Parametric solid stored in a file, into a directory of VRML files. This output directory contains assembly structure data, part and assembly names, and geometrical data representing the parts. This function accepts as input only Creo Parametric assemblies or parts.

ProExportVRML() supports creation of multiple output files from either parts or assemblies. If you export an assembly, the function creates an output file for each member of the assembly and one for the assembly itself. Default file names are:

```
asm1_a.wrl, asm2_a.wrl, ... asmN_a.wrl
```

where asm is the assembly name.

If you export parts, ProExportVRML() creates an output file for each part. Default names are part_p.wrl, where part is the part name. For more information on ProExportVRML(), refer to “Exporting Files to VRML” or “Batch Utilities” in the “Interface” section of Creo Parametric help.

The functions ProProductviewexportoptsAlloc() and ProProductviewexportoptsFree() allocate and free the memory assigned to the ProProductviewExportOptions object containing the Creo View export formats.

The function ProProductviewexportoptsFormatSet() assigns the flag specifying the Creo View export format.

The function ProProductviewFormattedMdlnameExport() exports a part or an assembly to one of the following user-defined Creo View formats.

- PRO_PV_FORMAT_PVS
- PRO_PV_FORMAT_ED
- PRO_PV_FORMAT_EDZ
- PRO_PV_FORMAT_PVZ

Example 2: To Export a Model File to IGES Format

The sample code in UgInterfaceExport.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_interface shows how to export a model file to IGES format using options similar to those seen in the UI.

Export Options in the JT Configuration File

You can export up to three levels of detail (LODs) from Creo Parametric files to the JT format. An LOD is a graphical representation of details such as the chord height, angle control, and step size for faceted surfaces. Each faceted surface is made up of a specific number of triangles. These shaded triangles represent the object. The more triangles that describe the object, the more details you can view.

You can create and use a configuration file, jt.config, to control the export of the LODs from Creo Parametric files to the JT format. The jt.config file is located in the <creo_loadpoint>\<datecode>\Common Files\text\jt\ directory. The options set in this file override the configuration options in the config.pro configuration file.

You can include the following parameters or export options in the jt.config file:

- EAITranslator—Specifies the setting for the export of control parameters.
  - chordalOption—Specifies whether to apply the chordal value as an absolute value in model units or as a relative value that is a percentage of the part size. The valid values are:
    - RELATIVE—Tessellates all the parts in a model relative to their size equally. The chordal value is applied as a fractional percentage.

        - ABSOLUTE—Tessellates all the parts in a model regardless of their size. The chordal value is applied as an absolute value.
    - structureOption—Specifies the mapping of the JT product structure to the JT file structure in the JT files after export. The supported product structures are:
        - JtkPER_PART—Specifies that the assembly is exported as a single JT file. The parts in the assembly are exported and saved as individual JT files in a sub-directory of the same name as the top-level assembly file.
        - JtkFULL_SHATTER—Specifies that each component of the assembly is exported as a separate JT file.
        - JtkMONOLITHIC—Specifies that the assembly is exported as a single JT file.
    - writeWhichFiles—Specifies which components must be exported to JT format.
        - ALL—This is the default option. Specifies that the entire assembly along with its parts must be exported.
        - ASSEMBLY_ONLY—Specifies that only the product structure must be exported.
        - PARTS_ONLY—Specifies that only the parts in the assembly must be exported without the assembly structure.
    - JtFileFormat—Specifies the version of the JT format in which the files must be exported. Refer to the Creo Parametric Data Exchange Help, for more information on AUTO option.
    - includeBrep—Specifies a boolean value to include the geometry boundary representation definition, that is, JT-brep or XT-brep, in the files exported to JT format. It controls the export of annotations as semantic representations. You cannot include the JT-brep and the XT-brep data structures in the same file. The valid values are true or false.
    - autoXtBrep—Specifies a boolean value to automatically convert the boundary representation geometry to parasolid format during the export. When you set the option to true, the boundary representation data is converted to the parasolid format. When set to false, the boundary representation data is stored in the proprietary JT format.

      You can use the autoXtBrep option with the includeBrep option to switch between the JT-brep and the XT-brep structures.
    - numLODs—Specifies the number of LOD definitions. PTC recommends creating up to three LODs.
- LOD—Specifies a group of parameters that control tessellation for a specific LOD. It also specifies the number of the current LOD.
    - Level—Specifies the current LOD number.

- `Chordal`—Specifies the maximum distance that a tessellated line segment deviates from the actual curve it approximates. It takes the value as a floating number from [0.0,1.0]. The value specified is as determined by the `chordalOption` option. If the chordal value exceeds the model size, you can consider the model size as the chordal value.

  Note

  For best results, use chordal values in conjunction with the `Angular` parameter. Chordal values primarily affect the larger features of the model while the angular values affect the smaller features of the model.

- `Length`—Specifies the maximum absolute length of the tessellated line segments in a curve approximation. If the `Length` value exceeds the model size, consider the default value as (`model_size/30`).
- `Label`—Specifies the user-defined name for the LOD.
- `Angular`—Specifies the angle control value for LOD definitions and triangulation export to JT format. This parameter sets the absolute maximum angle between two adjacent line segments in a curve approximation. The angle has its value between 0 and 90 degrees. The maximum angle value indicates coarse tessellation while the lowest value indicates fine tessellation and high quality LOD.

- `proeConfig`—Specifies the options that are specific to Creo Parametric.
  - `autoLODgeneration`—Specifies a boolean value to automatically generate three LODs from Creo Parametric. The valid values are `true` or `false`.

    Note

    When `autoLODgeneration` is set to `true`, the LOD options set in the `jt.config` file are ignored.

- autoLODStepSize—Specifies a boolean value to use the step size parameter during the automatic generation of LODs using the options set in Creo Parametric. This option is available only when autoLODgeneration is set to true.

  Note

  When autoLODStepSize is set to true, the step size set in the jt.config file is ignored.

- LOD[n]angle—Regulates the amount of additional improvement along curves with small radii for LOD1, LOD2, and LOD3. It takes the value as a floating number from [0.0,1.0]. This option is available only when autoLODgeneration is set to false.

- UseJTAngularControl—Specifies the type of angle control to be used. Angles can be controlled using the Creo Parametric angle control values or the JT angle control values. Set this parameter to true to use the JT angle control values. Specify the angle control values in the Angular option.

  By default, this parameter is set to false. In this case, the Creo Parametric angle control options, LOD1angle, LOD2angle, LOD3angle, and so on, are used to define the LOD generation.

You must set the configuration options intf3d_out_jt_config_name and intf3d_out_jt_config_path to specify the name and location of the jt.config file.

Shrinkwrap Export

To improve performance in large assembly design, you can export lightweight representations of models called Shrinkwrap models. A shrinkwrap model is based on the external surfaces of the source part or assembly model and captures the outer shape of the source model.

You can create the following types of non associative exported Shrinkwrap models:

- Surface Subset—This type consists of a subset of the original model's surfaces.
- Faceted Solid—This type is a faceted solid representing the original solid.
- Merged Solid—The external components from the reference assembly model are merged into a single part representing the solid geometry in all collected components.

Export Format	Creo Parametric TOOLKIT Functions	Type Constant
Shrinkwrap	`ProSolidShrinkwrap Create()`	N/A

Function Introduced:

- **ProSolidShrinkwrapMdlnameCreate()**

You can export the specified solid model as a Shrinkwrap model using the function `ProSolidShrinkwrapMdlnameCreate()`. This function requires:

- The model to be exported as Shrinkwrap
- The template model where the Shrinkwrap geometry will be created.
- The name of the exported file if the export format is VRML or STL.

Setting Shrinkwrap Options

Functions Introduced:

- **ProShrinkwrapoptionsAlloc()**
- **ProShrinkwrapoptionsFree()**
- **ProShrinkwrapoptionsQualitySet()**
- **ProShrinkwrapoptionsAutoholefillingSet()**
- **ProShrinkwrapoptionsIgnoreskeletonsSet()**
- **ProShrinkwrapoptionsIgnorequiltsSet()**
- **ProShrinkwrapoptionsIgnoreconstrbodiesSet()**
- **ProShrinkwrapoptionsAssignmasspropsSet()**
- **ProShrinkwrapoptionsDatumrefsSet()**

The function `ProShrinkwrapoptionsAlloc()` allocates memory for the structure defining the shrinkwrap options. The types of shrinkwrap methods are:

- `PRO_SWCREATE_SURF_SUBSET`—Surface Subset
- `PRO_SWCREATE_FACETED_SOLID`—Faceted Solid
- `PRO_SWCREATE_MERGED_SOLID`—Merged Solid

The function returns the options handle which is used to set the members of the structure defining the shrinkwrap options.

The function `ProShrinkwrapoptionsFree()` frees the memory allocated by the function `ProShrinkwrapoptionsAlloc()`.

The function `ProShrinkwrapoptionsQualitySet()` specifies the quality level for the system to use when identifying surfaces or components that will contribute to the Shrinkwrap model. Quality ranges from 1 which produces the coarsest representation of the model in the fastest time, to 10 which produces the most exact representation. The default value is true.

The function `ProShrinkwrapoptionsAutoholefillingSet()` sets a flag that forces Creo Parametric to identify all holes and surfaces that intersect a single surface and fills those holes during shrinkwrap. The default value is true.

The function `ProShrinkwrapoptionsIgnoreskeletonsSet()` determines whether the skeleton model geometry must be included in the Shrinkwrap model.

`ProShrinkwrapoptionsIgnorequiltsSet()` determines whether external quilts will be included in the Shrinkwrap model.

`ProShrinkwrapoptionsIgnoreconstrbodiesSet()` determines whether construction bodies are included in the Shrinkwrap model.

`ProShrinkwrapoptionsAssignmasspropsSet()` assigns mass properties to the Shrinkwrap model. The default value is false and the mass properties of the original model is assigned to the Shrinkwrap model. If the value is set to true, the user will have to assign a value for the mass properties.

`ProShrinkwrapoptionsDatumrefsSet()` selects the datum planes, points, curves, axes, and coordinate system references to be included from the Shrinkwrap model.

Surface Subset Options

Functions Introduced:

- **ProShrinkwrapoptionsIgnoresmallsurfsSet()**
- **ProShrinkwrapoptionsAdditionalsurfacesSet()**

The function `ProShrinkwrapoptionsIgnoresmallsurfsSet()` sets a flag that forces Creo Parametric to skip surfaces smaller than a certain size. The default value of this argument is false. The size of the surface is specified as a percentage of the model's size.

Use `ProShrinkwrapoptionsAdditionalsurfacesSet()` to select individual surfaces to be included in the Shrinkwrap model.

Faceted Solid Options

Functions Introduced:

- **ProShrinkwrapoptionsFacetedformatSet()**
- **ProShrinkwrapoptionsFramesFileSet()**

Use the function `ProShrinkwrapoptionsFacetedformatSet()` to specify the output file format of the Shrinkwrap model. The types of output format are:

- `PRO_SWFACETED_PART`——Creo Parametric part with normal geometry. This is the default format type.
- `PRO_SWFACETED_LIGHTWEIGHT_PART`—Lightweight Creo Parametric part with lightweight, faceted geometry.
- `PRO_SWFACETED_STL`—An STL file
- `PRO_SWFACETED_VRML`—A VRML file

The function `ProShrinkwrapoptionsFramesFileSet()` enables you to select a frame file to create a faceted solid motion envelope model that represents the full motion of the mechanism captured in the frame file. Specify the name and complete path of the frame file.

Merged Solid Options

Function Introduced:

- **ProShrinkwrapoptionsAdditionalcomponentsSet()**

Use the function `ProShrinkwrapoptionsAdditionalcomponentsSet()` to select individual components of the assembly to be merged into the Shrinkwrap model.

Example 3: To Export a Model to VRML Format

The sample code in `UgInterfaceExport.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_interface` shows how to create a faceted shrinkwrap model in VRML format

Example 4: To Create a Shrinkwrap Part Model as a Merged Solid

The sample code in `UgInterfaceExport.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_interface` shows how to create a new empty model and copy the merged solid shrinkwrap information into it.

Exporting to PDF and U3D

The functions described in this section support the export of Creo Parametric drawings and solid models to Portable Document Format (PDF) and U3D format. You can export a drawing or a 2D model as a 2D raster image embedded in a PDF file. You can export Creo Parametric solid models in the following ways:

- As a U3D model embedded in a one-page PDF file
- As 2D raster images embedded in the pages of a PDF file representing saved views
- As a standalone U3D file

While exporting multiple sheets of a Creo Parametric drawing to a PDF file, you can choose to export all sheets, the current sheet, or selected sheets.

These functions also allow you to insert a variety of non-geometric information to improve document content, navigation, and search.

Functions Introduced:

- **ProPDFoptionsAlloc()**
- **ProPDFoptionsIntpropertySet()**
- **ProPDFoptionsBoolpropertySet()**
- **ProPDFoptionsStringpropertySet()**
- **ProPDFoptionsDoublepropertySet()**
- **ProPDFExport()**
- **ProPDFWithProfileExport()**
- **ProPDFoptionsFree()**

The function `ProPDFoptionsAlloc()` allocates memory for the PDF options structure.

The function `ProPDFoptionsIntpropertySet()` sets the value for an integer or enum property of the PDF options structure. The types of export options are as follows:

- `PRO_PDFOPT_EXPORT_MODE`—Enables you to select the object to be exported to PDF and the export format. The values are:
  - `PRO_PDF_2D_DRAWING`—Specifies that only 2D drawings will be exported to PDF. This is the default.
  - `PRO_PDF_3D_AS_NAMED_VIEWS`—Specifies that Creo Parametric models will be exported as 2D raster images embedded in PDF files.
  - `PRO_PDF_3D_AS_U3D_PDF`—Specifies that Creo Parametric models will be exported as U3D models embedded in one-page PDF files.

- PRO_PDF_3D_AS_U3D—Specifies that a Creo Parametric model will be exported as a U3D (*.u3d) file. This value ignores the options available for the PDF options structure.

- PRO_PDFOPT_PDF_SAVE—Enables you to specify the PDF format while exporting 2D drawings and solid models. The values are:
  - PRO_PDF_ARCHIVE_1—Exports the 2D drawing to the PDF/A format. This type is applicable only for 2D drawings, that is, when you set the option PRO_PDFOPT_EXPORT_MODE to PRO_PDF_2D_DRAWING.

    If you set the option PRO_PDFOPT_PDF_SAVE to PRO_PDF_ARCHIVE_1, the following options are set as shown below:
    - PRO_PDFOPT_LAYER_MODE is set to PRO_PDF_LAYERS_NONE.
    - PRO_PDFOPT_HYPERLINKS is set to FALSE, that is, hyperlinks are not created in the PDF.
    - Shaded views will not be transparent and may overlap other data.
    - PRO_PDFOPT_PASSWORD_TO_OPEN is set to NULL.
    - PRO_PDFOPT_MASTER_PASSWORD is set to NULL.
    - PRO_PDF_FULL—Exports the object to the standard PDF format. This is the default value.
- PRO_PDFOPT_FONT_STROKE—Enables you to switch between using TrueType fonts in the resulting document and drawing or "stroking" text as line segments. The values are:
  - PRO_PDF_USE_TRUE_TYPE_FONTS—Specifies TrueType fonts. This is the default.
  - PRO_PDF_STROKE_ALL_FONTS—Specifies the option to stroke all fonts.
- PRO_PDFOPT_COLOR_DEPTH—Enables you to choose between color, grayscale, or monochrome output. The values are:
  - PRO_PDF_CD_COLOR—Specifies color output. This is the default.
  - PRO_PDF_CD_GRAY—Specifies grayscale output.
  - PRO_PDF_CD_MONO—Specifies monochrome output.
- PRO_PDFOPT_HIDDENLINE_MODE—Enables you to set the style for hidden lines in the resulting PDF document. The values are:
  - PRO_PDF_HLM_DASHED—Specifies dashed hidden lines. This is the default.
  - PRO_PDF_HLM_SOLID—Specifies solid hidden lines.

- PRO_PDFOPT_RASTER_DPI—Enables you to set the resolution for the output of any shaded views in DPI. The value is restricted to the values in ProDotsPerInch, and the default is PRORASTERDPI_300.
- PRO_PDFOPT_LAYER_MODE—Enables you to set the availability of layers in the document. The values are:
  - PRO_PDF_LAYERS_VISIBLE—Exports only visible layers in a drawing.
  - PRO_PDF_LAYERS_NONE—Exports only the visible entities in the drawing, but not the layers on which they are placed.
  - PRO_PDF_LAYERS_ALL—Exports the visible layers and entities. This is the default.
- PRO_PDFOPT_PARAM_MODE—Enables you to set the availability of model parameters as searchable metadata in the PDF document. The values are:
  - PRO_PDF_PARAMS_DESIGNATED—Exports only the specified model parameters in the PDF metadata.
  - PRO_PDF_PARAMS_NONE—Exports the drawing to PDF without the model parameters.
  - PRO_PDF_PARAMS_ALL—Exports the drawing and the model parameters to PDF. This is the default.
- PRO_PDFOPT_ALLOW_MODE—Defines the changes that you can make in the PDF document. This option can be set only if PRO_PDFOPT_RESTRICT_OPERATIONS is set to true. The permitted viewer operations are given by the following values:
  - PRO_PDF_RESTRICT_NONE—This is the default value and it restricts you from performing all operations in the PDF document.
  - PRO_PDF_RESTRICT_FORMS_SIGNING—Allows you to fill in the fields of the form, create templates, and add digital signatures to the PDF document.
  - PRO_PDF_RESTRICT_INSERT_DELETE_ROTATE—Allows you to insert, delete, and rotate pages in the PDF document.
  - PRO_PDF_RESTRICT_COMMENT_FORM_SIGNING—Allows you to add or edit comments, fill in the fields of the form, create templates, and add digital signatures to the PDF document.
  - PRO_PDF_RESTRICT_EXTRACTING—Allows you to perform all viewer operations, except for extracting pages from the PDF document.
- PRO_PDFOPT_ALLOW_PRINTING_MODE—Allows you to set the print resolution. This option can be set only if the options PRO_PDFOPT_

RESTRICT_OPERATIONS and PRO_PDFOPT_ALLOW_PRINTING are set to true. The values are:

  - PRO_PDF_PRINTING_LOW_RES—Specifies low resolution for printing.
  - PRO_PDF_PRINTING_HIGH_RES—Specifies high resolution for printing. This is the default.

- PRO_PDFOPT_LINECAP—Enables you to control the treatment of the ends of the geometry lines exported to PDF. The values are:
  - PRO_PDF_LINECAP_BUTT—Specifies the butt cap square end. This is the default.
  - PRO_PDF_LINECAP_ROUND—Specifies the round cap end.
  - PRO_PDF_LINECAP_PROJECTING_SQUARE—Specifies the projecting square cap end.
- PRO_PDFOPT_LINEJOIN—Enables you to control the treatment of the joined corners of connected lines exported to PDF. The values are:
  - PRO_PDF_LINEJOIN_MITER—Specifies the miter join. This is the default.
  - PRO_PDF_LINEJOIN_ROUND—Specifies the round join.
  - PRO_PDF_LINEJOIN_BEVEL—Specifies the bevel join.
- PRO_PDFOPT_SHEETS—Enables you to specify the sheets from a Creo Parametric drawing that are to be exported to PDF. The values are:
  - PRINT_CURRENT_SHEET—Specifies that only the current sheet will be exported to the PDF file.
  - PRINT_ALL_SHEETS—Specifies that all the sheets will be exported to the PDF file. This is the default.
  - PRINT_SELECTED_SHEETS—Specifies that sheets of a specified range will be exported to the PDF file. If this value is assigned, then the value of the string property PRO_PDFOPT_SHEET_RANGE must also be included.
- PRO_PDFOPT_LIGHT_DEFAULT—Enables you to set the default lighting style used while exporting Creo Parametric models in the U3D format to a one-page PDF file. The values are:
  - PRO_PDF_U3D_LIGHT_NONE—Specifies no lights.
  - PRO_PDF_U3D_LIGHT_WHITE—Specifies white lights.
  - PRO_PDF_U3D_LIGHT_DAY—Specifies day lights.
  - PRO_PDF_U3D_LIGHT_BRIGHT— Specifies bright lights.
  - PRO_PDF_U3D_LIGHT_PRIMARY—Specifies primary color lights.

- PRO_PDF_U3D_LIGHT_NIGHT—Specifies night lights.
- PRO_PDF_U3D_LIGHT_BLUE—Specifies blue lights.
- PRO_PDF_U3D_LIGHT_RED—Specifies red lights.
- PRO_PDF_U3D_LIGHT_CUBE—Specifies cube lights.
- PRO_PDF_U3D_LIGHT_CAD—Specifies CAD optimized lights. This is the default value.
- PRO_PDF_U3D_LIGHT_HEADLAMP—Specifies headlamp lights.

• PRO_PDFOPT_RENDER_STYLE_DEFAULT—Enables you to set the default rendering style used while exporting Creo Parametric models in the U3D format to a one-page PDF file. The values are:

- PRO_PDF_U3D_RENDER_BOUNDING_BOX—Specifies bounding box rendering.
- PRO_PDF_U3D_RENDER_TRANSPARENT_BOUNDING_BOX—Specifies transparent bounding box rendering.
- PRO_PDF_U3D_RENDER_TRANSPARENT_BOUNDING_BOX_OUTLINE—Specifies transparent bounding box outline rendering.
- PRO_PDF_U3D_RENDER_VERTICES—Specifies vertices rendering.
- PRO_PDF_U3D_RENDER_SHADED_VERTICES—Specifies shaded vertices rendering.
- PRO_PDF_U3D_RENDER_WIREFRAME—Specifies wireframe rendering.
- PRO_PDF_U3D_RENDER_SHADED_WIREFRAME—Specifies shaded wireframe rendering.
- PRO_PDF_U3D_RENDER_SOLID—Specifies solid rendering. This is the default.
- PRO_PDF_U3D_RENDER_TRANSPARENT—Specifies transparent rendering.
- PRO_PDF_U3D_RENDER_SOLID_WIREFRAME—Specifies solid wireframe rendering.
- PRO_PDF_U3D_RENDER_TRANSPARENT_WIREFRAME—Specifies transparent wireframe rendering.
- PRO_PDF_U3D_RENDER_ILLUSTRATION—Specifies illustrated rendering.
- PRO_PDF_U3D_RENDER_SOLID_OUTLINE—Specifies solid outlined rendering.
- PRO_PDF_U3D_RENDER_SHADED_ILLUSTRATION—Specifies shaded illustrated rendering.

- PRO_PDF_U3D_RENDER_HIDDEN_WIREFRAME—Specifies hidden wireframe rendering.

- PRO_PDFOPT_SIZE—Enables you to specify the page size of the exported PDF file. The values are restricted to the value of ProPlotPaperSize. If the value is VARIABLE_SIZE_IN_MM_PLOT or VARIABLE_SIZE_PLOT, the size must be specified in the PRO_PDFOPT_HEIGHT and PRO_PDFOPT_WIDTH properties.
- PRO_PDFOPT_ORIENTATION—Enables you to specify the orientation of the pages in the exported PDF file. The values are:
  - PRO_ORIENTATION_PORTRAIT—Exports the pages in portrait orientation. This is the default.
  - PRO_ORIENTATION_LANDSCAPE—Exports the pages in landscape orientation.

  The option PRO_PDFOPT_ORIENTATION is not available if the property PRO_PDFOPT_SIZE is set to VARIABLE_SIZE_IN_MM_PLOT or VARIABLE_SIZE_PLOT
- PRO_PDFOPT_VIEW_TO_EXPORT—Enables you to specify the view or views to be exported to the PDF file. The values are:
  - PRO_PDF_VIEW_SELECT_CURRENT—Exports the current graphical area to a one-page PDF file.
  - PRO_PDF_VIEW_SELECT_BY_NAME—Exports the selected view to a one-page PDF file with the view name printed at the bottom center of the view port. If this value is assigned, then the value of the string property PRO_PDFOPT_SELECTED_VIEW must also be included.
  - PRO_PDF_VIEW_SELECT_ALL—Exports all the views to a multi-page PDF file. Each page contains one view with the view name displayed at the bottom center of the view port.
- PRO_PDFOPT_INCL_ANNOT—Enables you to specify if annotations must be included when Creo Parametric models are exported as U3D graphics in a PDF file. The values are:
  - PRO_PDF_INCLUDE_ANNOTATION—Includes annotations when models are exported.
  - PRO_PDF_EXCLUDE_ANNOTATION—Excludes annotations when models are exported. This is the default value.

The function ProPDFoptionsBoolpropertySet() sets the value for a boolean property of the PDF options structure. The types of export options are as follows:

- PRO_PDFOPT_SEARCHABLE_TEXT—If true, stroked text is searchable. The default value is true.
- PRO_PDFOPT_LAUNCH_VIEWER—If true, launches the Adobe Acrobat Reader. The default value is true.
- PRO_PDFOPT_HYPERLINKS—Sets Web hyperlinks to be exported as label text only or sets the underlying hyperlink URLs as active. The default value is true, specifying that the hyperlinks are active.
- PRO_PDFOPT_BOOKMARK_ZONES—If true, adds bookmarks to the PDF showing zoomed in regions or zones in the drawing sheet. The zone on an A4-size drawing sheet is ignored.
- PRO_PDFOPT_BOOKMARK_VIEWS—If true, adds bookmarks to the PDF document showing zoomed in views on the drawing.
- PRO_PDFOPT_BOOKMARK_SHEETS—If true, adds bookmarks to the PDF document showing each of the drawing sheets.
- PRO_PDFOPT_BOOKMARK_FLAG_NOTES—If true, adds bookmarks to the PDF document showing the text of the flag note.
- PRO_PDFOPT_RESTRICT_OPERATIONS—If true, allows you to restrict or limit operations on the PDF document using the ProPDFRestrictOperationsMode modification flags. The default is false.
- PRO_PDFOPT_ALLOW_PRINTING—If true, allows you to print the PDF document. The default value is true.
- PRO_PDFOPT_ALLOW_COPYING—If true, allows you to copy content from the PDF document. The default value is true.
- PRO_PDFOPT_ALLOW_ACCESSIBILITY—If true, enables visually-impaired screen reader devices to extract data independent of the value of the enum ProPDFRestrictOperationsMode. The default value is true.
- PRO_PDFOPT_PENTABLE—If true, uses the standard Creo Parametric pentable to control the line weight, line style, and line color of the exported geometry. The default value is false.
- PRO_PDFOPT_PENTAB_FOR_TEXT—If true, the standard Creo Parametric pentable is used to control the thickness of the stroked text of the exported geometry. If false, the stroked text will be exported with their original thickness and the thickness value defined by the pentable will be ignored. The default value is true.
- PRO_PDFOPT_ADD_VIEWS—If true, allows you to add view definitions to the U3D model from a file. The default value is true.

The function ProPDFoptionsStringpropertySet() sets the value for a string property of the PDF options structure. The types of export options are as follows:

- PRO_PDFOPT_TITLE—Specifies a title for the PDF document.
- PRO_PDFOPT_AUTHOR—Specifies the name of the person generating the PDF document.
- PRO_PDFOPT_SUBJECT—Specifies the subject of the PDF document.
- PRO_PDFOPT_KEYWORDS—Specifies relevant keywords in the PDF document.
- PRO_PDFOPT_PASSWORD_TO_OPEN—Sets a password to open the PDF document. If the value is not set or NULL, anyone can open the PDF document without a password.
- PRO_PDFOPT_MASTER_PASSWORD—Sets a password to restrict or limit the viewer operations that you can perform on the opened PDF document. If the value is set to NULL, you can make any changes to the PDF document regardless of the settings of the PRO_PDFOPT_ALLOW_* modification flags.
- PRO_PDFOPT_SHEET_RANGE—Specifies the range of sheets in a Creo Parametric drawing that are to be exported to a PDF file. If this property is assigned, then the integer property PRO_PDFOPT_SHEETS is set to the value PRINT_SELECTED_SHEETS.
- PRO_PDFOPT_SELECTED_VIEW—Sets the option PRO_PDFOPT_VIEW_TO_EXPORT to the value PRO_PDF_VIEW_SELECT_BY_NAME, if the corresponding view is successfully found.

The function ProPDFoptionsDoublepropertySet() sets the value for a double property of the PDF options structure. The types of export options are as follows:

- PRO_PDFOPT_HEIGHT—Enables you to set the height for a user-defined page size of the exported PDF file. The default value is 0.0. This option is available only if the enum PRO_PDFOPT_SIZE is set to VARIABLE_SIZE_IN_MM_PLOT or VARIABLE_SIZE_PLOT.
- PRO_PDFOPT_WIDTH—Enables you to set the width for a user-defined page size of the exported PDF file. The default value is 0.0. This option is available only if the enum PRO_PDFOPT_SIZE is set to VARIABLE_SIZE_IN_MM_PLOT or VARIABLE_SIZE_PLOT.
- PRO_PDFOPT_TOP_MARGIN—Enables you to specify the top margin of the view port. The default value is 0.0.
- PRO_PDFOPT_LEFT_MARGIN—Enables you to specify the left margin of the view port. The default value is 0.0.

- PRO_PDFOPT_BACKGROUND_COLOR_RED—Enables you to specify the default red background color that appears behind the U3D model. You can set any value within the range of 0.0 through 255.0. The default value is 255.0.
- PRO_PDFOPT_BACKGROUND_COLOR_GREEN—Enables you to specify the default green background color that appears behind the U3D model. You can set any value within the range of 0.0 through 255.0. The default value is 255.0.
- PRO_PDFOPT_BACKGROUND_COLOR_BLUE—Enables you to specify the default blue background color that appears behind the U3D model. You can set any value within the range of 0.0 through 255.0. The default value is 255.0.

The function ProPDFExport() exports the file to a PDF document based on the export settings defined in the PDF options structure. Specify the complete name and path, including the extension of the output file.

Use the function ProPDFWithProfileExport() to repeatedly export 2D drawings to the PDF format with the same export options. These options are stored in an XML file called a profile. You can have several such profiles. The input arguments to this function are:

- *model*—A drawing model to export. This drawing model must be open and the drawing window must be active.
- *output_file*—The complete path to the output file with extension.
- *profile*—The path to the profile to be used.

Use the function ProPDFoptionsFree() to free the memory contained in the PDF options structure.

Importing Parameter Files

Functions Introduced:

- **ProInputFileRead()**

The function ProInputFileRead() imports files of several types to create data in Creo Parametric. The file types are declared in ProUtil.h. The import formats and their type constants are as listed in the following table:

Import Format	Creo Parametric TOOLKIT Functions	Type Constant
Relations file	ProInputFileRead()	PRO_RELATION_FILE
Program file		PRO_PROGRAM_FILE
Configuration options file		PRO_CONFIG_FILE
Setup file		PRO_DWG_SETUP_FILE
Spool file		PRO_SPOOL_FILE
Cable Parameters file		PRO_CABLE_PARAMS_FILE
Connector Parameters file		PRO_CONNECTOR_PARAMS_FILE
Model Tree Configuration		PRO_ASSEM_TREE_CFG_FILE

Import Format	Creo Parametric TOOLKIT Functions	Type Constant
file		
Wirelist file		PRO_WIRELIST_FILE
SLD Variant file		SLD_VARIANT_FILE

The option PRO_RELATION_FILE reads a text file that contains a list of all the model relations and parameters relations in exactly the same format as the Creo Parametric user enters them.

Use the function ProInputFileRead() with the argument PRO_CONNECTOR_PARAMS to identify the connectors. To access parameters on connectors and their entry ports use the following arguments:

- *arg1*—Represents the integer pointer to ProIdTable. ProIdTable is an integer array of component identifiers.
- *arg2*—Represents the integer pointer to the number of component identifiers.

Use the function ProInputFileRead() with the argument PRO_CABLE_PARAMS_FILE to read cable parameters. You need to set the following arguments:

- *arg1*—Represents a ProSolid (part pointer).
- *arg2*—Represents the cable name.

Use the function ProInputFileRead() with the argument PRO_WIRELIST_FILE to read files in Mentor Graphics LCABLE format. This function does not create wires, but provides parameters from a wire list for use when creating in a harness assembly a wire with the same name as that in the LCABLE file.

Use the function ProInputFileRead() with the argument PRO_RELATION_FILE to get the individual feature relations. To access feature relations use the following arguments:

- *arg2*—Represents the individual feature relations. It is an integer pointer to the feature identifier that gets the relations contained in a feature. If this is NULL you get the relations contained in the model.
- *arg3*—It is an integer pointer. If it points to 1, then the relations in the file must be added to the current relations, otherwise the relations in the file must replace the current relations.

Use function ProInputFileRead() with argument PRO_SPOOL_FILE to create new spools or update existing ones.

Importing 2D Models

Import Format	Creo Parametric TOOLKIT Functions	Type Constant
STEP file	`Pro2dImportMdlnameCreate()`, `Pro2dImportAppend()`	`PRO_STEP_FILE`
IGES (2D) file		`PRO_IGES_FILE`
DXF file		`PRO_DXF_FILE`
DWG file		`PRO_DWG_FILE`
CGM file		`PRO_CGM_FILE`
MEDUSA file		`PRO_MEDUSA_FILE`
Creo Elements/Direct drafting files (`.mi`, `.bi`, and `.bdl`)		`PRO_CCD_DRAWING_FILE`
IGES (2D) file	`ProInputFileRead()`	`PRO_IGES_SECTION`

Functions Introduced:

- **Pro2dImportMdlnameCreate()**
- **Pro2dImportAllSheets()**
- **Pro2dImportAppend()**
- **ProInputFileRead()**

The function `Pro2dImportMdlnameCreate()` imports interface files and creates a new two-dimensional model with the specified name. The created models can be drawings, layouts, diagrams, drawing formats. Use the argument to control whether or not to import two-dimensional views.

If you want to import all the drawing sheets for formats that support multiple drawing sheets, use the function `Pro2dImportAllSheets()`. The function imports interface files with all the drawing sheets, and creates a new two-dimensional model. For the model type `PRO_MDL_DWGFORM`, only the first two drawing sheets are imported.

The function `Pro2dImportAppend()` appends a two-dimensional model to the specified model.

You can import and append the Creo Elements/Direct drafting files using the functions `Pro2dImportMdlnameCreate()` and `Pro2dImportAppend()`.

Use the function `Pro2dImportMdlnameCreate()` to import the Creo Elements/Direct drafting file to a Creo Parametric drawing file. The following data is imported from the `.mi`, `.bi`, and `.bdl` format files:

- Basic entities such as point, line, arc, fillet, circle, polygon, text, spline, b-spline, center line, symmetry line, reference text line, and projected reference points
- 2D construction geometry such as lines, arcs, circles, splines, and so on
- Linear, angular, radial, diameter, ordinate, and leader dimensions

- Tolerance values in dimensions
- Leader and non-leader notes
- Annotation views
- Drawing sheets, symbol, views, and layers
- The attributes of color, layers, line types, and text fonts
- All types of hatch including user-defined hatches

You can set the mapping options in the mapping file `mi_import.pro`. The entity attributes such as color, line type, text fonts, leader arrow styles and so on are imported from Creo Elements/Direct drafting files to Creo Parametric drawing files depending on the settings in the mapping file.

Refer to the Creo Parametric Data Exchange Help for more information on importing Creo/Elements Direct drafting files in Creo Parametric drawing file.

Use the function `ProInputFileRead()` with the argument `PRO_IGES_SECTION` to import a 2D IGES section into a sketch.

Importing 3D Models

The functions described in this section are used to import files of different format types into Creo Parametric.

Functions Introduced:

- **ProIntfimportSourceTypeGet()**
- **ProIntfimportModelWithOptionsMdlnameCreate()**
- **ProIntfimportLayerFilter()**

Import Format	Creo Parametric TOOLKIT Functions	Type Constant
ACIS file	`ProIntfimportModelWithOptionsMdlnameCreate()`	`PRO_INTF_IMPORT_ACIS`
CADDS file		`PRO_INTF_IMPORT_CADDS`
CATIA (.model) file		`PRO_INTF_IMPORT_CATIA_MODEL`
CATIA (.session) file		`PRO_INTF_IMPORT_CATIA_SESSION`
DXF file		`PRO_INTF_IMPORT_DXF`
ICEM file		`PRO_INTF_IMPORT_ICEM`
IGES file		`PRO_INTF_IMPORT_IGES`
Neutral file		`PRO_INTF_IMPORT_NEUTRAL`
Parasolid-based CADDS system file		`PRO_INTF_IMPORT_PARASOLID`
POLTXT file		`PRO_INTF_IMPORT_POLTXT`
STEP file		`PRO_INTF_IMPORT_STEP`
VDA file		`PRO_INTF_IMPORT_VDA`
CATIA (.CATpart) file		`PRO_INTF_IMPORT_CATIA_PART`

Import Format	Creo Parametric TOOLKIT Functions	Type Constant
UG file		PRO_INTF_IMPORT_UG
Creo View (.ol and .ed) files		PRO_INTF_IMPORT_ PRODUCTVIEW
JT Open format		PRO_INTF_IMPORT_JT
CATIA Graphical Representation (CGR) format		PRO_INTF_IMPORT_CATIA_CGR
SolidWorks Part (.sldprt) file		PRO_INTF_IMPORT_SW_PART
SolidWorks Asembly (.sldasm) file		PRO_INTF_IMPORT_SW_ASSEM
Inventor Part (.ipt) file		PRO_INTF_IMPORT_INVENTOR_ PART
Inventor Assembly (.iam) file		PRO_INTF_IMPORT_INVENTOR_ ASSEM
STL file		PRO_INTF_IMPORT_STL
VRML file		PRO_INTF_IMPORT_VRML
CATIA (.product) file		PRO_INTF_IMPORT_CATIA_ PRODUCT
Creo Elements/Direct file (Assemblies and parts) • bundle—.bdl • modeling —soliddesigner .sda, .sdp,.sdac, and .sdpc • package—.pkg		PRO_INTF_IMPORT_CC
Solid Edge Part (.par) file		PRO_INTF_IMPORT_SEDGE_ PART
Solid Edge Assembly (.asm) file		PRO_INTF_IMPORT_SEDGE_ ASSEMBLY
Solid Edge Sheet metal (.psm) file		PRO_INTF_IMPORT_SEDGE_ SHEETMETAL_PART
3D Manufacturing Format (3MF)		PRO_INTF_IMPORT_3MF

The following data is included during the import of models from other formats to Creo Parametric:

- 3D Manufacturing Format (3MF)—From Creo Parametric 5.0.1.0 onward, you can import 3MF files containing part and assembly models to Creo Parametric. You can import part-level colors, top-assembly parameters, and facet geometry from 3MF models to Creo Parametric.
- Autodesk Inventor—You can import Autodesk Inventor models to Creo Parametric. The import includes basic geometry such as solids, quilts, and surfaces from Autodesk Inventor models to Creo Parametric. You can also import datum features, colors, attributes, and wire body datum curves from Inventor part and assembly models.

  Note

  Depending on the Autodesk Inventor model, Creo Parametric imports the model as a part or an assembly. To let Creo Parametric decide if the Autodesk Inventor model must be imported as a part or assembly, in the function `ProIntfimportModelWithOptionsMdlnameCreate()`, you must specify the input argument `ProMdlType` as `PRO_MDL_UNUSED`.

- JT—JT models are imported to Creo Parametric with their color overrides. Components with color overrides at any level in an assembly structure are supported.

  From Creo Parametric 3.0 onward, the Product Manufacturing Information (PMI) of the annotations is imported as semantic representation from JT models to Creo Parametric models. The semantic import is supported only for 3D notes and basic dimensions. All the other types of annotations are imported as graphical entities. You can import the planar and zonal cross-sections of part and assembly models from JT files to Creo Parametric.

  Note

  From Creo Parametric 2.0 M200 onward, the license `INTF_for_JT` is required to import a JT file to Creo Parametric. If the license is not available the functions return the error `PRO_TK_NO_LICENSE`.

- Creo Elements/Direct—From Creo Parametric 3.0 onward, the Product Manufacturing Information (PMI) of the annotations is imported as semantic representation fromCreo Elements/Direct models to Creo Parametric models. The semantic import is supported only for 3D notes and basic dimensions. All

the other types of annotations are imported as graphical entities. You can also import the clipping features owned by the Creo Elements/Direct part and assembly models as cross-sections in Creo Parametric.

- Creo View—You can import colors assigned to the components of assemblies and their sub-assembly models, including the colors of the sub-level entities such as parts, quilts, and faces from Creo View to Creo Parametric. Creo View models are imported to Creo Parametric with their color overrides. Components with color overrides at any level in an assembly structure are supported. Along with components, color overrides are also supported for component model items, such as, face and quilts.
- SolidWorks—You can import basic geometry such as solids, quilts, and surfaces from SolidWorks models to Creo Parametric. The import includes datum features, colors, attributes, and layers.

Note

Depending on the SolidWorks model, Creo Parametric imports the model as a part or an assembly. To let Creo Parametric decide if the SolidWorks model must be imported as a part or assembly, in the function `ProIntfimportModelWithOptionsMdlnameCreate()`, you must specify the input argument `ProMdlType` as `PRO_MDL_UNUSED`.

- Solid Edge—From Creo Parametric 3.0 M010 onward, you can import Solid Edge part and assembly models to Creo Parametric. The import includes boundary representation geometry, datum features, colors, and attributes.

  From Creo Parametric 3.0 M030 onward, Solid Edge models are imported to Creo Parametric with their color overrides. Components with color overrides at any level in an assembly structure are supported.

  From Creo Parametric 3.0 M020 onward, you can also import a Solid Edge sheet metal part to Creo Parametric.

Note

Depending on the Solid Edge model, Creo Parametric imports the model as a part or an assembly. To let Creo Parametric decide if the Solid Edge model must be imported as a part or assembly, in the function `ProIntfimportModelWithOptionsMdlnameCreate()`, you must specify the input argument `ProMdlType` as `PRO_MDL_UNUSED`.

- Unigraphics—You can import basic geometry such as solids, quilts, and surfaces from Unigraphics models to Creo Parametric. The import includes datum features, colors, attributes, and layers.

Note

Depending on the Unigraphics model, Creo Parametric imports the model as a part or an assembly. To let Creo Parametric decide if the Unigraphics model must be imported as a part or assembly, in the function `ProIntfimportModelWithOptionsMdlnameCreate()`, you must specify the input argument `ProMdlType` as `PRO_MDL_UNUSED`.

Note

Refer to the Creo Parametric Data Exchange Help for more information on importing geometry to Creo Parametric. Refer to the compatibility matrix on PTC.com for the supported software versions.

The function `ProIntfimportSourceTypeGet()` is a utility that returns the type of model that can be created from the geometry file. This function is not applicable for all formats. If this function is not valid for a geometric file, you will need to know the type of model you want to create (part, assembly, or drawing).

The function `ProIntfimportModelWithOptionsMdlnameCreate()` imports objects of other formats using a profile and creates a new model or set of models with the specified name and representation. Once the profile is set, it remains valid for the entire session unless it is reset with another profile. The input arguments of this function are:

- *import_file*—Full path to file to be imported.
- *profile*—The import profile path. An import profile is an XML file with the extension `.dip`. It contains the options that control an import operation. It also contains all the options for the supported 3D import formats. Refer to the

Creo Parametric Online Help for more information on creation and modification of import profiles.

Note

The input argument *profile* allows you to include the import of Creo Elements/Direct containers, face parts, wire parts, and empty parts.

- *type*—The type of file to be imported.

  The following formats are supported for importing structure and graphics level of details:

  - Creo View (i.e. Product View, .ol, .ed, .edz, .pvs, .pvz) files (`PRO_INTF_IMPORT_PRODUCTVIEW`)
  - CATIA V5 (CATPart (`PRO_INTF_IMPORT_CATIA_PART`, CATProduct (`PRO_INTF_IMPORT_CATIA_PRODUCT`), CGR (`PRO_INTF_IMPORT_CATIA_CGR`)) files
  - SolidWorks (.sldprt (`PRO_INTF_IMPORT_SW_PART`), .sldasm(`PRO_INTF_IMPORT_SW_ASSEM`)) files
  - Unigraphics NX (`PRO_INTF_IMPORT_UG`) files
- *create_type*—The type of model to create. This could be part, assembly, or drawing (for STEP associative drawings).
- *rep type*—The representation type to be used for importing. The enumerated type `ProImportRepType` defines the representation type and has the following values:

  - `PRO_IMPORTREP_MASTER`—This is the default import type. It imports the geometric and the nongeometric data (annotations, datums, coordinate systems, creation of features and so on) of the assembly and displays the full representation of the assembly.
  - `PRO_IMPORTREP_STRUCTURE`—Imports the product structure or the meta data of the assemblies.
  - `PRO_IMPORTREP_GRAPHICS`—Imports the display data of the assemblies.

Note

The import representation types `PRO_IMPORTREP_STRUCTURE` and `PRO_IMPORTREP_GRAPHICS` do not create model geometry in Creo Parametric, although they allow the import of the fully-functional assemblies.

- *new_model_name*— The name of the new top level import model.
- *filter_func*—Callback to the function `ProIntfimportLayerFilter()` that determines how to display and map layers from the imported model. If this is `NULL`, the default layer handling will take place.
- *application_data*—The application data to be passed to the filter function. Can be `NULL`.
- *created_model*—The handle to the created model (in case of an assembly – the handle to the top assembly). Even if this is NULL, the model is created.

Note

- When importing an assembly using the function `ProIntfimportModelWithOptionsMdlnameCreate()` if any component of the assembly is missing, then during import an empty placeholder with the missing component name is created in the model tree. For example, during import if the missing component is a part, an empty part is created. Similarly, if the missing component is a subassembly, then an empty subassembly is created. Placeholders for missing components are created only for the following formats:
  - CATIA V5
  - CATIA V4
  - Unigraphics
  - CADDS 5
  - SolidWorks
  - Creo View (.ol and .ed) files
  - JT Open format
- The functions `ProIntfimportModelCreate()` and `ProIntfimportModelWithOptionsCreate()` have been deprecated.

  The function `ProIntfimportModelWithOptionsMdlnameCreate()` supersedes the functions `ProIntfimportModelCreate()` and `ProIntfimportModelWithOptionsCreate()`. The deprecated functions may not be supported in future releases of Creo Parametric It is recommended that you rebuild your applications with the new function.

  Use the function `ProIntfimportModelWithOptionsMdlnameCreate()` instead with *profile* as NULL and *rep type* (representation type) as `PRO_IMPORTREP_MASTER`.

- From Creo Parametric 3.0 M080 onward, the function `ProIntfimportModelWithOptionsMdlnameCreate()` imports a JT file to Creo Parametric only if the license `INTF_for_JT` is available. If the license is not available the functions return the error `PRO_TK_NO_LICENSE`.
- From Creo Parametric 2.0 M200 onward, the functions `ProIntfimportModelCreate()`, `ProIntfimportModelWithOptionsCreate()` and `ProIntfimportModelWithProfileCreate()` import a JT file to Creo Parametric only if the license `INTF_for_JT` is available. If the license is not available the functions return the error `PRO_TK_NO_LICENSE`.

The function `ProIntfimportModelWithProfileCreate()` imports objects of other formats using a profile and creates a new model or set of models with the specified name.

The function `ProIntfimportModelWithProfileCreate()` has been deprecated. Use the function `ProIntfimportModelWithOptionsMdlnameCreate()` instead with `PRO_IMPORTREP_MASTER` as the representation type.

The function `ProIntfimportLayerFilter()` is a callback function that allows your application to determine the status of each of the imported layers. You can modify the layer information using the functions described in the next section.

Modifying the Imported Layers

Layers help you organize model items so that you can perform operations on those items collectively. These operations primarily include ways of showing the items in the model, such as displaying or blanking, selecting, and suppressing. The methods described in this section modify the attributes of the imported layers.

Functions Introduced:

- **ProLayerfilterdataNameGet()**
- **ProLayerfilterdataNameSet()**
- **ProLayerfilterdataCountsGet()**
- **ProLayerfilterdataActionSet()**

Imported layers are identified by their names. The function `ProLayerfilterdataNameGet()` returns the name of the layer while the function `ProLayerfilterdataNameSet()` can be used to set the name of the layer.

The function `ProLayerfilterdataCountsGet()` specifies the following:

- Number of curves on the specified layer
- Number of surfaces on the specified layer
- Number of trimmed surfaces on the specified layer

The function `ProLayerfilterdataActionSet()` sets the display status of the imported layers. You can set the display status of the layers to one of the following:

- Show—Display the specified layer.
- Blank—Make the specified layer blanked.
- Hidden—(Assembly mode only) Make the specified layer hidden.
- Skip—Do not import the entities on this layer.
- Ignore—Import only entities on this layer but not the layer

Example 5: Importing a 3D Model With Layer Filter Options

The sample code in `UgInterfaceImport.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_interface` shows how to import from an IGES file while filtering out a certain layer to be renamed and blanked, using the filter function `UserLayerFilter()`. It also demonstrates how to use the function `ProIntfimportSourceTypeGet()` to determine the type of the model.

Validation Score for Imports

Functions Introduced:

- **ProIntfimportValidationscoreCalculate()**
- **ProIntfimportValidationscoreGet()**
- **ProIntfimportValidationpreferencesLoad()**

You can import CATIA V5, NX, SolidWorks, SolidEdge, Autodesk Inventor, and Creo View files with various properties such as surface area, volume, and the coordinates for the center of gravity (COG) into Creo Parametric. After import, Creo Parametric has to calculate the value of these properties and report if there are any differences in the values. Also general failures during import, such as, failure to import an assembly component or failure to solidify a part must be reported.

To report the value differences and failures, the validation score for a model is created. The validation score is created for imported, regenerated, or updated models. The properties such as, surface area, volume, and the coordinates for the center of gravity (COG) are converted in to geometrical validation properties. Assembly models include the validation property that is related to the product structure. Creo Parametric compares the validation properties of the imported

models with the mass properties of the source models and computes a score of the data conversion. The validation score of a model `PTC_VAL_IMP_SCORE` is a model parameter. The score is created for the parent assembly and its individual components and has the following values:

- `PASS`
- `PASS_WITH_WARNINGS`
- `FAIL`

Use the function `ProIntfimportValidationscoreCalculate()` to calculate the validation score for the specified model. The function `ProIntfimportValidationscoreGet()` retrieves the validation score for the specified model, where the score is already calculated using the function `ProIntfimportValidationscoreCalculate()`.

The validation score depends on the data conversion from source files to Creo Parametric. If you want certain parameters to be ignored and passed, you can set these preferences for validation data in the `config.val` file. Use the function `ProIntfimportValidationpreferencesLoad()` to specify the `config.val` file that must be used to load the preferences for data validation while calculating validation score.

Refer to the Creo Parametric Data Exchange online help, for more information.

29

Interface: Importing Features

This chapter describes how to create import features in Creo Parametric.

Note

PTC strongly recommends that you create import features in a Creo Parametric session that has only those models open where the feature is to be created. It is also recommended not to follow the creation of import features with calls to the functions `ProMdlErase()` or `ProMdlEraseAll()` until the control returns to Creo Parametric, since this may interfere with erasing temporary objects at the end of the import.

Creating Import Features from Files

To create import features in Creo Parametric from external format files use the functions described in this section.

Functions Introduced:

- **ProIntfDataSourceInit()**
- **ProImportfeatCreate()**
- **ProImportfeatureWithProfileCreate()**

Superseded Functions

- **ProImportfeatWithProfileCreate()**

The function `ProIntfDataSourceInit()` is used to build the interface data source required by the functions `ProImportfeatCreate()` and `ProImportfeatureWithProfileCreate()`.

The input arguments of this function are:

- *intf_type*—Specifies the type of file to import. The valid format files from which the user can create the import features are specified in the enumerated data type `ProIntfType`:

Type Constant	Import Format
`PRO_INTF_NEUTRAL_FILE`	Neutral file
`PRO_INTF_IGES`	IGES 3D file
`PRO_INTF_STEP`	STEP file
`PRO_INTF_VDA`	VDA file
`PRO_INTF_SET`	SET file
`PRO_INTF_PDGS`	PDGS file
`PRO_INTF_ICEM`	ICEM file
`PRO_INTF_ACIS(*.sat)`	ACIS format file
`PRO_INTF_DXF`	DXF file
`PRO_INTF_CDRS`	CDRS file
`PRO_INTF_STL`	STL file
`PRO_INTF_VRML`	VRML file
`PRO_INTF_PARASOLID`	Parasolid-based CADDS system file
`PRO_INTF_AI`	AI file
`PRO_INTF_CATIA_PART`	CATIA (.CATpart) file
`PRO_INTF_UG`	UG file
`PRO_INTF_PRODUCTVIEW`	Creo View (.ol) files
`PRO_INTF_CATIA_PRODUCT`	CATIA V5 Assembly file
`PRO_INTF_CATIA_CGR`	CATIA Graphical Representation (CGR) format
`PRO_INTF_JT`	JT Open Interface
`PRO_INTF_INVENTOR_PART`	Inventor Part (.ipt) file
`PRO_INTF_INVENTOR_ASM`	Inventor Assembly (.iam) file

Type Constant	Import Format
PRO_INTF_SE_PART	Solid Edge part (.par) file
PRO_INTF_SE_SHEETMETAL_PART	Solid Edge Sheet metal (.psm) file
PRO_INTF_3MF	3D Manufacturing Format (3MF)

- *p_source*—the name of the file with extension. The specified format file should exist in the current working directory or in a path specified in the `search_path` configuration option.

This function returns the handle to the `ProIntfDataSource` object, which should be passed to the functions `ProImportfeatCreate()` and `ProImportfeatureWithProfileCreate()`.

The function `ProImportfeatCreate()` is used to create a new import feature in the Creo Parametric solid model. The input arguments of this function are:

- *p_solid* —Specifies the part in which the user wants to create the import feature.
- *data_source*—Specifies a pointer to the interface data source. Use the function `ProIntfDataSourceInit()` to get the handle to the `ProIntfDataSource` object.
- *p_csys* — Specifies the coordinate system of the part with which the user wants to align the import feature. If this is `NULL`, the function uses the default coordinate system in the Creo Parametric model and the import feature will be aligned with respect to this coordinate system.
- *p_attributes*—Specifies the attributes for the creation of the new import feature. Please see the section Import Feature Attributes on page 736 for more information.

Note

From Creo Parametric 2.0 M200 onward, the function `ProImportfeatCreate()` imports a JT file to Creo Parametric only if the license `INTF_for_JT` is available. If the license is not available the function returns the error `PRO_TK_NO_LICENSE`.

The function `ProImportfeatCreate()` returns the `ProFeature` handle for the created import feature.

In Creo Parametric 7.0.0.0 and later, the function `ProImportfeatWithProfileCreate()` is deprecated. Use the function `ProImportfeatureWithProfileCreate()` instead.

The function ProImportfeatureWithProfileCreate() is used to create a new import feature in the Creo Parametric solid model. The input arguments follow:

- *p_solid*—Pointer to the solid part. Assembly case is not supported.
- *data_source*—Source of data to create the import feature.
- *p_csys*—Pointer to the reference coordinate system. If this is NULL, the function uses the default coordinate system.
- *profile*—Path to the import file. If this value is NULL, the function ProImportfeatureWithProfileCreate() works same as ProImportfeatCreate().

Note

An import profile is an XML file with the extension dip (Dex In Profile) and contains the options that control an import operation. It contains all the options for the supported 3D import formats. Refer to the Creo Parametric Help for more information on creation and modification of import profiles.

- *cut_or_add*—Set to PRO_B_TRUE for imported geometry representing a cut or PRO_B_FALSE otherwise.
- *body_use_opt*—Generic body options.
- *body_arr*—ProArray of bodies. Size of ProArray must be 1.

The output argument p_feat_handle is the handle to the new import feature. If this is NULL, the feature is still created.

Note

- The function ProImportfeatureWithProfileCreate() (cannot create an import feature using an import profile for the STL and VRML formats. Once a profile is set, it remains valid for the entire session unless it is reset with another profile.
- From Creo Parametric 2.0 M200 onward, the function ProImportfeatureWithProfileCreate() imports a JT file to Creo Parametric only if the license INTF_for_JT is available. If the license is not available the function returns the error PRO_TK_NO_LICENSE.

The function ProDatumcurveFromfileCreate() creates a new import feature containing a datum curve in the Creo Parametric model. The input arguments of this function are:

- *p_solid*—Specifies the part in which you create the import feature.
- *full_file_name*—Specifies the name of the file from which you create the import feature.
- *file_type*—Specifies the file type to import. It is given by the `ProIntfType` enumerated type. The file types supported by this function are as follows.

Type Constant	Import Format
PRO_INTF_NEUTRAL_FILE	Neutral file
PRO_INTF_IGES	IGES 3D file
PRO_INTF_STEP	STEP file
PRO_INTF_VDA	VDA file
PRO_INTF_ACIS	ACIS format file
PRO_INTF_DXF	DXF file
PRO_INTF_AI	AI file
PRO_INTF_CATIA_PART	CATIA (.CATpart) file
PRO_INTF_UG	UG file
PRO_INTF_JT	JT Open Interface
PRO_INTF_IBL	IBL file
PRO_INTF_PTS	PTS file

  For all other file types that are not supported, the function `ProDatumcurveFromfileCreate()` returns `PRO_TK_INVALID_TYPE`.
- *p_csys* — Specifies the coordinate system of the part with which you align the import feature If this is `NULL`, the function uses the default coordinate system and the import feature is aligned with respect to this coordinate system.

Note

From Creo Parametric 2.0 M200 onward, the function `ProDatumcurveFromfileCreate()` creates a new import JT feature only if the license `INTF_for_JT` is available. If the license is not available the function returns the error `PRO_TK_NO_LICENSE`.

Creating Import Features from Arbitrary Geometric Data

You can create an import feature in a Creo Parametric model by building the required entity data in the Creo Parametric TOOLKIT application.

The advantages of importing features from a Creo Parametric TOOLKIT application are:

- You can create virtually non-parametric user-defined geometry at a desired location. This is sometimes an alternative to parametric feature creation, which can be more complicated.
- Import features are regenerated more quickly than corresponding groups of parametric features.
- You can integrate Creo Parametric with non-Creo Parametric supported geometry file formats.

The following sequence of steps is required to create the import feature from memory:

- Allocate the interface data.
- Add surfaces, edges, quilts, and datums.
- Create the import feature from the interface data.

These steps are described in detail in the following sections.

Allocating ProInterfacedata

Function Introduced:

- **ProIntfDataAlloc()**

Use the function `ProIntfDataAlloc()` to allocate memory for the interface data structure.

Adding Surfaces

Function Introduced:

- **ProSurfacedataAlloc()**

Use the function `ProSurfacedataAlloc()` to allocate memory for the surface data. Once the surface data is initialized, it will be appended to the interface data.

Initializing Surface Data

Functions Introduced:

- **ProSurfacedataInit()**
- **ProPlanedataInit()**
- **ProCylinderdataInit()**
- **ProConedataInit()**

- **ProTorusdataInit()**
- **ProSrfrevdataInit()**
- **ProTabcyldataInit()**
- **ProRulsrfdataInit()**
- **ProSplinesrfdataInit()**
- **ProCylsplsrfdataInit()**
- **ProBsplinesrfdataInit()**
- **ProFilsrfdataInit()**

Use the function `ProSurfacedataInit()` to initialize the surface data structure.

The input arguments of this function are:

- *Surface_type*—Specifies the type of surface to be created. The types of surfaces are:
  - `PRO_SRF_PLANE`—Plane
  - `PRO_SRF_CYL`—Cylinder
  - `PRO_SRF_CONE`—Cone
  - `PRO_SRF_TORUS`—Torus
  - `PRO_SRF_COONS`—Coons Patch
  - `PRO_SRF_SPL`—Spline Surface
  - `PRO_SRF_FIL`—Fillet Surface
  - `PRO_SRF_RUL`—Ruled Surface
  - `PRO_SRF_REV`—General Surface of Revolution
  - `PRO_SRF_TABCYL`—Tabulated Cylinder
  - `PRO_SRF_B_SPL`—B-spline surface
  - `PRO_SRF_FOREIGN`—Foreign Surface
  - `PRO_SRF_CYL_SPL`—Cylindrical Spline Surface

  The type of the surface determines the function to be used to initialize the surface data structure.For example, if the type of surface to be created is `PRO_SRF_PLANE`, then the function `ProPlanedataInit()` should be used to initialize the surface data structure
- *surf_uv_min*—Specifies the minimum uv extents of the surface.
- *surf_uv_max*—Specifies the maximum uv extents of the surface.
- *surf_orient*—Specifies the orientation of the surface. By default the value is `PRO_SURF_ORIENT_OUT`

- *p_surf_shape*—The data containing the information about the shape of the surface.
- *Surface_Id*—Specifies a unique identifier of the Surface.

Depending on the shape of the surface, call one of the following functions to create the surface data structure `ProSurfaceshapedata` and assign it to variable `p_surf_shape` of function `ProSurfacedataInit()`. Ensure that the function used to create the `ProSurfaceshapedata` matches with the `ProSrftype` value used in `ProSurfacedataInit()`.

- `ProPlanedataInit()`
- `ProCylinderdataInit()`
- `ProConedataInit()`
- `ProTorusdataInit()`
- `ProSrfrevdataInit()`
- `ProTabcyldataInit()`
- `ProRulsrfdataInit()`
- `ProSplinesrfdataInit()`
- `ProCylsplsrfdataInit()`
- `ProBsplinesrfdataInit()`

Note

Set the configuration option `intf_in_keep_high_deg_bspl_srfs` to `YES` to preserve the B-spline surfaces returned by `ProBsplinesrfdataInit()` in the `ProIntData` data structure. If this configuration option is not set, these surfaces are interpreted as spline surfaces.

- `ProFilsrfdataInit()`

Refer to the Geometry Representations on page 2147 appendix for more information on how to use the above functions.

Note

The following return values for the functions `ProBsplinesrfdataInit()`, `ProRulsrfdataInit()`, `ProSrfrevdataInit()`, and `ProTabcyldataInit()` should be treated as warnings:

`PRO_TK_BSPL_UNSUITABLE_DEGREE`

`PRO_TK_BSPL_NON_STD_END_KNOTS`

`PRO_TK_BSPL_MULTI_INNER_KNOTS`

They indicate that the geometry finally imported in Creo Parametric is different from the geometry initially supplied to the above functions. The geometry is not rejected by the functions and is used to generate the `ProSurfaceshapedata` data structure.

Surfacedata Contours

The geometric representation of the surface created above is unbounded, that is the nature of the surface boundaries is determined by its array of contours. Multiple contours can be used for surfaces with internal voids.

Functions Introduced:

- **ProSurfacedataContourArraySet()**
- **ProContourdataAlloc()**
- **ProContourdataInit()**
- **ProContourdataEdgeIdArraySet()**

Use the function `ProSurfacedataContourArraySet()` to set an array of contours on the surface.

The input arguments of this function are:

- *p_surf_data*—Specifies the surface data to which the array of contour data is to be set.
- *contour_array*—Specifies an array of contours on the surface. The `ProContourdata` handle can be obtained by using the following functions in sequence:

  `ProContourdataAlloc()`

  `ProContourdataInit()`

  `ProContourdataEdgeIdArraySet()`

Use the function `ProContourdataAlloc()` to allocate memory to the contour data structure.

Use the function `ProContourdataInit()` to initialize the contour data structure. The input argument of this function is:

- *contour_trav* — Specifies the contour traversal. This parameter has the following values:
  - `PRO_CONTOUR_TRAV_INTERNAL`—Internal Contour
  - `PRO_CONTOUR_TRAV_EXTERNAL`—External Contour

The function returns the allocated contour data structure.

Use the function `ProContourdataEdgeIdArraySet()` to set identifiers to an array of edges, that form the boundary of the specified surface.

The input arguments of this function are:

- *p_contour_data*—Specifies the contour data to which the array of edge identifiers have to be set.
- *edge_id_arr*—Specifies the array of edge identifiers. These identifiers must be same as those provided in the `ProEdgedata` structures described below.

  For example, if the surface is bounded by 4 edges, then the identifier of each edge should be assigned to each element of an array of integers of size 4.

Appending the Surface Data to the Interface Data

Function Introduced:

- **ProIntfDataSurfaceAppend()**

Use the function `ProIntfDataSurfaceAppend()` to append the surface data into the interface data.

Repeat the sequence for each surface desired in the import feature.

Adding Edges

Functions Introduced:

- **ProEdgedataAlloc()**
- **ProEdgedataInit()**
- **ProCurvedataAlloc()**
- **ProLinedataInit()**
- **ProArcdataInit()**
- **ProEllipsedataInit()**

- **ProSplinedataInit()**
- **ProBsplinedataInit()**

If the import feature to be created requires any edge information, then call the functions list above in sequence, else skip this section.

Use the function `ProEdgedataAlloc()` to allocate memory for the edge data structure. After initialization, this data will be appended to the interface data.

Use the function `ProEdgedataInit()` to initialize the edge data structure. The following are the input arguments:

- *edge_id* — Specifies a unique identifier of the edge.
- *edge_surf_ids*—Specifies the ID of the surfaces on either side of the edge.
- *edge_directions* —Specifies the edge directions on the surface.
- *edge_uv_point_arr* —Specifies an array of UV points on the surfaces. The value can be `NULL`.
- *p_edge_uv_curve_data*—Specifies the edge UV curves on the surfaces. The value can be `NULL`.
- *p_edge_curve_data*—Specifies the curve data handle in the form of the `ProCurvedata` structure. This data handle is returned by the functions `ProLinedataInit()`, `ProArcdataInit()`, `ProEllipsedataInit()`, `ProSplinedataInit()`, or `ProBsplinedataInit()`. Use the function `ProCurvedataFree` to free the `ProCurvedata` data handle.

Note

PTC recommends that you split the closed loop edge into two or more continuous edges while specifying the inputs to the function `ProEdgedataInit()`. For example, to create a circular edge, instead of specifying the start angle as 0 and the end angle as 360, split the circular edge into 2 or more edges. The angular measurements of the split edges could be 0 to 30 for the first split and 30 to 360 for the second split. The function `ProEdgedataInit()` must be called for each split.

Use the function `ProCurvedataAlloc()` to allocate memory for the curve data structure. The curve data structure defines the edge profile.

Depending on the type of curve specified for the edge, call one of the following functions to initialize the curve data.

- `ProLinedataInit()`
- `ProArcdataInit()`

- `ProEllipsedataInit()`
- `ProSplinedataInit()`
- `ProBsplinedataInit()`

Use the function `ProLinedataInit()` to initialize the line data structure. Specify the start of the line and end of the line as inputs of this function.

Use the function `ProArcdataInit()` to initialize an arc data structure. The input arguments of this function are:

- *vector1*—Specifies the first vector of the arc coordinate system.
- *vector2*—Specifies the second vector of the arc coordinate system.
- *origin*—Specifies the center of the arc coordinate system
- *start_angle*—Specifies the starting angle (in radians) of the arc.
- *end_angle* —Specifies the end angle (in radians) of the arc.
- *radius* —Specifies the radius of the arc.

Use the function `ProEllipsedataInit()` to initialize an ellipse data structure. The input arguments of this function are:

- *center*—Specifies the center of the ellipse.
- *x_axis*—Specifies the first (x) axis vector of the ellipse.
- *plane_normal*—Specifies the axis vector that is normal to the plane of the ellipse.
- *x_radius*—Specifies the radius of the ellipse in the direction of 'x' axis.
- *y_radius*—Specifies the radius of the ellipse in the direction of 'y' axis. The 'y' axis can be found as a vector product of the `plane_normal` on `x_axis`.
- *start_ang*—Specifies the starting angle (in radians) of the ellipse.
- *end_ang*—Specifies the end angle (in radians) of the ellipse.

Use the function `ProSplinedataInit()` to initialize the spline data structure. The input arguments of this function are:

- *par_arr* —Specifies an array of spline parameters
- *pnt_arr*—Specifies an array of spline interpolant points
- *tan_arr*—Specifies an array of tangent vectors at each point
- *num_points*—Specifies the size for all the arrays

Use the function `ProBsplinedataInit()` to initialize the B-spline data structure. The input arguments of this function are:

- *degree*—Specifies the degree of the basis function.
- *params* —Specifies an array of knots on the parameter line.

- *weights* —In the case of rational B-splines, it specifies an array of the same dimension as the array of *c_pnts*. Else, the value of this argument is `NULL`.
- *c_pnts*—Specifies an array of knots on control points.
- *num_knots*—Specifies the size of the params array.
- *num_c_points*—Specifies the size of the *c_pnts* and the size of weights if it is not `NULL`.

Note

Although `ProBsplinedataInit()` returns B-spline curves, these curves are interpreted as spline curves in the `ProIntData` data structure used by the function `ProImportfeatCreate()` while creating the import feature.

The values `PRO_TK_BSPL_UNSUITABLE_DEGREE` and `PRO_TK_BSPL_NON_STD_END_KNOTS` returned by `ProBsplinedataInit()` should be treated as warnings. These values indicate that the geometry finally imported in Creo Parametric is different from the geometry initially supplied to the function. The geometry is not rejected by `ProBsplinedataInit()` and is used to generate the `ProCurvedata` data structure.

Appending the Edge Data to the Interface Data

Function Introduced:

- **ProIntfDataEdgeAppend()**

Use the function `ProIntfDataEdgeAppend()` to append the edge data into the interface data.

Repeat the sequence for each edge required by the import feature.

Adding Quilts

Functions Introduced:

- **ProQuiltdataAlloc()**
- **ProQuiltdataInit()**
- **ProQuiltdataSurfArraySet()**
- **ProIntfDataQuiltAppend()**

Use the function `ProQuiltdataAlloc()` to allocate memory to the quilt data structure.

Use the function `ProQuiltdataInit()` to assign the user defined identity to the quilt data structure. Specify a unique identity for the quilt as the input argument. The function returns the handle to the quilt data structure.

Use the function `ProQuiltdataSurfArraySet()` to define an array of surfaces as a quilt. The input arguments of this function are:

- *p_quilt_data*—Specifies a handle to the quilt data to which we want to assign the set of surfaces.
- *arr_p_surf*—Specifies an array of surfaces that will be defined as a quilt.

Use the function `ProIntfDataQuiltAppend()` to append the quilt data to the interface data. The input arguments of this function are:

- *p_intfdata* —Specifies a handle to the interface data to which you want to append the quilt data.
- *p_quiltdata* —Handle to the quilt data.

Repeat the sequence for each quilt required in the import feature.

Adding Datums

Functions Introduced:

- **ProDatumdataAlloc()**

Use the function `ProDatumdataAlloc()` to allocate memory to the datum data structure.

Initializing Datums

- **ProDatumdataInit()**
- **ProDatumCsysdataInit()**
- **ProDatumCurvedataInit()**
- **ProDatumPlanedataInit()**
- **ProDatumdataMemoryFree()**

Use the function `ProDatumdataInit()` to initialize the datum data structure. The input arguments of this function are:

- *datum_id*—Specifies a unique identifier of the datum.
- *datum_type*—Specifies the datum type. The types of datums are:
  - `PRO_CSYS`
  - `PRO_CURVE`
  - `PRO_DATUM_PLANE`

- *datum_name*—Specifies the name to be assigned to the datum.
- *p_datum_obj*—The datum object that contains the geometrical information about the datum. Depending on the type of the datum to be created, one of the following functions must be used to create the `ProDatumobject` data structure.
  - `ProDatumCsysdataInit()`
  - `ProDatumCurvedataInit()`
  - `ProDatumPlanedataInit()`

> **Note**
>
> The value `PRO_TK_BSPL_MULTI_INNER_KNOTS` returned by `ProDatumCurvedataInit()` should be treated as a warning. This value indicates that the geometry finally imported in Creo Parametric is different from the geometry initially supplied to the function. The geometry is not rejected by `ProDatumCurvedataInit()` and is used to generate the `ProCurvedata` data structure.

Use the function `ProDatumdataMemoryFree()` to free the top-level memory used by the datum data structure.

Appending the Datum Data to the Interface Data

Use the function `ProIntfDataDatumAppend()` to append the datum data to the interface data required to create the import feature. The input arguments are:

- *p_intfdata*—Specifies the interface data to which the datum data must be appended.
- *p_datumdata* —Specifies a handle to the datum data obtained from the function `ProDatumdataInit()`.
- Repeat the sequence for each datum member required to be in the import feature.

Creating Features from the Interface Data

Functions Introduced:

- **ProIntfDataSourceInit()**
- **ProImportfeatCreate()**
- **ProIntfDataFree()**

Use the function `ProIntfDataSourceInit()` to build the interface data source required by the functions `ProImportfeatCreate()` and `ProImportfeatureWithProfileCreate()`. The input arguments of this function are:

- *intf_type*—Specifies the type of the interface. Since the user builds all the data required by the interface, the value should be `PRO_INTF_NEUTRAL`.
- *p_source*—Specifies the handle to the interface data source.

The function returns the handle `ProIntfDataSource`, which must be passed to the functions `ProImportfeatCreate()` and `ProImportfeatureWithProfileCreate()`.

Use the function `ProImportfeatCreate()` to create the import feature in the Creo Parametric solid model. The input arguments of this function are:

- *p_solid*—Specifies the part or assembly in which the user wants to create the import feature.
- *data_source*—Specifies a pointer to the interface data source. Use the function `ProIntfDataSourceInit()` to get the handle to the interface data source.
- *p_csys*—Specifies the co-ordinate system of the part with which you want to align the import feature. If this is `NULL`, the function uses the default coordinate system in the Creo Parametric model and the import feature will be aligned with respect to that coordinate system.
- *p_attributes*—Specifies the attributes for the creation of the new import feature. Refer to the section Import Feature Attributes on page 736 for more information.

The function `ProImportfeatCreate()` returns the `ProFeature` handle for the created import feature.

If your Creo Parametric TOOLKIT application is built using Creo Parametric 6.0 and you have the function `ProImportfeatCreate()` in your application, then the same application works as it is in Creo Parametric 7.0.0.0.

Note

If you are planning to recompile your Creo Parametric TOOLKIT in Creo Parametric 7.0.0.0, you must set the default value of the enumerated data type `body_use_opt` to `PRO_IMPORT_BODY_USE_DEFAULT`, `add_bodies` to `0`, and `body_arr` to `NULL` otherwise the behavior of Creo Parametric may be unpredictable.

The function `ProImportfeatureWithProfileCreate()` is used to create a new import feature in the Creo Parametric solid model. This function takes the same input arguments as the function `ProImportfeatCreate()`, except for the argument *profile* that specifies the path to the import profile used. An import profile is an XML file with the extension `.dip` (Dex In Profile) and contains the options that control an import operation. It contains all the options for the supported 3D import formats. Refer to the Creo Parametric Online Help for more information on creation and modification of import profiles.

Note

The function `ProImportfeatureWithProfileCreate()` cannot create an import feature using an import profile for the STL and VRML formats.

Use the function `ProIntfDataFree()` to release the memory occupied by the interface data.

Import Feature Attributes

Attributes define the action to be taken when creating the import feature. Following are the defined attributes:

- `attempt_make_solid`—Specifies whether the import feature is to be created as a solid or a surface type. Set the value to `1` to create an import feature of solid type. Set it to `0` to create a surface type of import feature.

  Note

  If the import feature is an open surface, setting `attempt_make_solid` to `1` does not make the import feature of solid type.

- `cut_or_add`—Specifies whether the solid type of import feature is to be created as a cut or a protrusion. This argument is valid only if `attempt_make_solid` is set to `1`. Set the value to `1` to cut the solid import feature from the intersecting solid. Set it to `0` to create it as a protrusion.

  Note

  When `attempt_make_solid` is set to `0`, the value assigned to `cut_or_add` is not considered.

- `join_surfaces`—Specifies whether the import feature is created as a single quilt (joined surface) or separate surfaces (as it was in the original file) if it is of surface type. This argument is valid only if `attempt_make_solid` is set to `0`. If the value is set to `1`, all surfaces that can be joined are joined to form a single quilt.
- `add_bodies`— Creates the same body structure as is present in the source file.
- `body_use_opt`—Specifies the body options you can use while importing a feature and is defined by the enumerated data type `ProImportBodyUseOpts`. The valid values are as follows:
  - `PRO_IMPORT_BODY_USE_DEFAULT`—Imports feature in the default body.
  - `PRO_IMPORT_BODY_USE_NEW`—Imports feature in a new body.
  - `PRO_IMPORT_BODY_USE_ALL`—Currently not supported.
  - `PRO_IMPORT_BODY_USE_SELECTED`—Imports feature in a selected body.
- `body_arr`—`ProArray` of bodies to be selected. By default, the size is `1`. Set this value as `NULL` if you do not want to use any bodies in the import operation.

Redefining the Import Feature

Use the following functions in sequence to redefine the import feature.

Functions Introduced:

- **ProImportfeatRedefSourceInit()**
- **ProImportfeatRedefine()**

Use the function `ProImportfeatRedefSourceInit()` to initialize the redefine source. Currently Creo Parametric TOOLKIT users may

- Redefine the attributes of any import feature.

> **Note**
>
> When redefining the attributes of the import feature, Creo Parametric will not use the value of the attribute `join_surfaces`, because this attribute is valid only for import feature creation.

- Redefine the geometry of an import feature created from a geometric file. Import features created from memory may not be redefined.

The input arguments are:

- *operation*—Specifies the type of operation to use when redefining the import feature.
- *p_source*—Specifies the handle to the new interface data or the new attributes structure.

The function `ProImportfeatRedefSourceInit()` returns the handle to a structure, that is passed as an input argument to the function `ProImportfeatRedefine()`.

Use the function `ProImportfeatRedefine()` to redefine the import feature. The input arguments are:

- *p_feat_handle*—Specifies the handle for the import feature to be redefined.
- *p_source*—The handle to be used for redefinition from the function `ProImportfeatRedefSourceInit()`.

> **Note**
>
> `ProImportfeatRedefine()` does not support ATB-enabled features. It returns `PRO_TK_BAD_CONTEXT` while accessing such features.

Import Feature Properties

Functions Introduced:

- **ProImportfeatIdArrayCreate()**
- **ProImportfeatIdArrayMapCount()**
- **ProImportfeatIdArrayMapGet()**
- **ProImportfeatIdArrayFree()**
- **ProImportfeatUserIdToItemId()**
- **ProImportfeatItemIdToUserId()**
- **ProImportfeatDataGet()**

Use the function `ProImportfeatIdArrayCreate()` to create an array of mappings between the user defined ids and the ids assigned by Creo Parametric to the entity items in the import feature.

Specify the handle to the feature, for which the user defined ids and ids assigned by Creo Parametric have to be mapped, as the input argument of the function. The function returns an array of mapped ids.

Use the function `ProImportfeatIdArrayMapCount()` to get the number of elements in the array of mappings.Use the function `ProImportfeatIdArrayMapGet()` to get the mapping of a particular element in the array.

Use the function `ProImportfeatIdArrayFree()` to free the array.

Use the function to `ProImportfeatUserIdToItemId()` to obtain the id or ids assigned by Creo Parametric for a user defined id. The function returns multiple ids in an array if the import operation split a particular entity.

For example, if you create a circular edge as a single edge data defined by a single id, Creo Parametric creates the circle by splitting it into two. If you pass the user defined id as an input to the function `ProImportfeatUserIdToItemId()`, the function will return an array of the ids assigned to each half of the circle.

The input arguments of this function are:

- *p_feat_handle*—Specifies the handle of the import feature.
- *user_id*—Specifies the identifier of the geometry item.
- *item_type*—Specifies the type of the geometry item. The types of geometry are:
  - `PRO_SURFACE`
  - `PRO_EDGE`
  - `PRO_QUILT`

Use the function `ProImportfeatItemIdToUserId()` to convert a Creo Parametric item id to an array of user defined ids.

For example, if the edges defined by the user are created as a single edge by Creo Parametric, and you pass a single item id assigned by Creo Parametric to the function `ProImportfeatItemIdToUserId()`, it will return an array of user ids.

- *p_feat_handle*—Specifies the handle of the import feature.
- *item_id*—Specifies the identifier of the geometry item.
- *item_type*—Specifies the type of the geometry item. The types of geometry are:
  - `PRO_SURFACE`
  - `PRO_EDGE`
  - `PRO_QUILT`

Use the function `ProImportfeatDataGet()` to retrieve the parameters assigned to the import feature. The output returned by this function will contains the following:

- Information about the interface type of the import feature.
- The filename from which the import feature is created. This is applicable for import features created from a file.
- The coordinate system with respect to which the import feature is aligned.
- The attributes of the import feature.

Extracting Creo Parametric Geometry as Interface Data

Functions Introduced:

- **ProPartToProInterfaceData()**
- **ProIntfDataAccuracyGet()**
- **ProIntfDataAccuracytypeGet()**
- **ProIntfDataOutlineGet()**
- **ProIntfDataDatumCount()**
- **ProIntfDataDatumGet()**
- **ProIntfDataEdgeCount()**
- **ProIntfDataEdgeGet()**
- **ProIntfDataQuiltCount()**
- **ProIntfDataQuiltGet()**
- **ProIntfDataSurfaceCount()**
- **ProIntfDataSurfaceGet()**

Superseded Functions:

- **ProPartToProIntfData()**

Use the function `ProPartToProIntfData()` to extract a `ProIntfData` structure describing the geometry of a part as if it were an import feature. This provides a single interface to extract all geometric data in order to convert it to another geometric format. The function `ProPartToProIntfData()` extracts the `ProIntfData` structure only if the model has a single body, else returns the error `PRO_TK_MULTIBODY_UNSUPPORTED`.

In Creo Parametric 7.0.0.0 and later, the function `ProPartToProIntfData()` has been deprecated. Use the function `ProPartToProInterfaceData()` to convert a `ProPart` structure to a `ProIntfData` structure. The input arguments follow:

- *ptk_part*—The `ProPart` data structure that needs to be converted.
- *p_cnv_opts*—The options for the conversion defined by the structure `ProPartConversionOptions`.

The interface data is returned by the output argument `p_intfdata`. You must preallocate the memory for this argument by calling the function `ProIntfDataAlloc()`.

The functions `ProIntfDataAccuracytypeGet()`, `ProIntfDataAccuracyGet()`, and `ProIntfDataOutlineGet()` provide access to properties of the interface data structure.

The other functions allow you to count and access each individual geometric data structure in the interface data.

Associative Topology Bus Enabled Interfaces

The following interfaces are ATB-enabled by default:

- CADDS 5
- CATIA V4
- CATIA V5
- Creo Elements/Direct
- Creo View
- Creo Granite file
- ICEM
- Neutral
- OPTEGRA
- SolidWorks
- NX
- Inventor

You can indicate in the import profile to disable ATB. You can do this in the **Import Profile Editor** (under **Tools ► Utilities** in the Creo Parametric user interface) by clearing the option **Enable ATB**.

Refer to the Creo Parametric Data Exchange Help for more information on profile editing and ATB (Associative Topology Bus).

Associative Topology Bus Enabled Models and Features

Associative Topology Bus (ATB) propagates changes made to the original CAD system data in the heterogeneous design environment. All geometric IDs preserved by the native system after the change to the native file are also preserved in the imported geometry by the ATB update. With ATB, you can work with Creo Parametric part or assembly that is:

- A Translated Image Model (TIM) representation of a model imported from the ATB interface, such as, CADDS or CATIA.
- A Creo Parametric assembly containing one or more components which are models imported from an ATB Interface, such as, CADDS or CATIA.
- A Creo Parametric part containing an Import Feature that is imported from an ATB Interface such as, ICEM.

Only import operations in Creo Parametric create TIM parts and assemblies. You can open CATIA, CADDS model files as TIMs. Neutral part files and files of other ATB-enabled formats are imported as native Creo Parametric parts with ATB-enabled features.

The TIM parts and assemblies store their ATB information at the model level. However, ATB-enabled import features store ATB information at the feature-level. The TIMs are displayed in the Model Tree with ATB icons that indicate their status with respect to their reference file as up-to-date, out-of-date, and so on.

The functions related to ATB models or features are available in the header file `ProATB.h`. These functions enable you to perform the following actions on a TIM model or ATB-enabled feature or the entire geometry of the imported model:

- Check the status of the TIMs or the ATB-enabled features.
- Update TIMs or ATB-enabled features that are identified as out-of-date.
- Change the link of a TIM or ATB-enabled feature.
- Break the association between a TIM or the ATB-enabled feature and the original reference model.

Functions Introduced:

- **ProModelIsTIM()**
- **ProModelHasTIMFeature()**
- **ProModelListTIMFeatures()**
- **ProATBMdlnameVerify()**
- **ProATBMarkAsOutOfDate()**
- **ProATBUpdate()**
- **ProATBRelink()**

The function `ProModelIsTIM()` checks if the specified model is a TIM.

The function `ProModelHasTIMFeature()` checks if the specified model contains a TIM feature.

The function `ProModelListTIMFeatures()` lists all the TIMs or ATB-enabled features present in the specified model. This function can be called after the function `ProModelHasTIMFeature()` which determines if the specified model has one or more TIM features.

The function `ProATBMdlnameVerify()` verifies if the specified ATB model is out of date with the source CAD model. The function first checks if the specified model is a TIM. If the model is not a TIM, this function checks if the ATB-enabled model was created by importing or appending ICEM or neutral surfaces to existing Creo Parametric part models. The input arguments for this function are:

- Model—Specify a Creo Parametric Part or Assembly that is—
  - A Translated Image Model (TIM) representation of a model imported from the ATB interface, such as, CADDS or CATIA.
  - A Creo Parametric assembly containing one or more components which are models imported from an ATB Interface, such as, CADDS or CATIA.
  - A Creo Parametric part containing an Import Feature that is imported from an ATB Interface such as, ICEM.
- feat_ids—Specify an array of feature ids for the ATB-enabled features in the model. If a model contains more than one ATB-enabled feature, the verify function works only on the specified feature. If you do not specify a feature id, the `ProATBMdlnameVerify()` function verifies the entire model including TIMs from non-native CAD models.
- search_paths—Specify the complete location to the source CAD model. You can specify multiple directories to search for the model. If no search path is specified, then the function will search in current working directory or locations set in config-option `atb_search_path`.

The output arguments of this function represent the status of the TIMs and are as follows:

- models_out_of_date—Specifies an array of TIMs or the ATB-enabled features that are out-of-date with the source model and require an update. These models are represented by a red icon in the Model Tree in the Creo Parametric user interface.
- models_unlinked—Specifies an array of TIMs or the ATB-enabled features

that have missing links because the reference model is missing from the designated search path. These models are represented by a yellow icon in the Model Tree in the Creo Parametric user interface.

- models_old_version—Specifies an array of TIMs for which the source CAD model is older than the one with which the TIM was last updated. These models are represented by a yellow icon in the Model Tree in the Creo Parametric user interface. Use the function `ProArrayFree()` to free the array of output arguments.

The function `ProATBMarkAsOutOfDate()` identifies all the ATB-enabled features that are out of date for the update operation.

The function `ProATBUpdate()` updates only those ATB-enabled models or features that are displayed in the session. The update action synchronizes the derived structure and the contents of the TIMs with the primary structure and the content of the source non-native CAD models. This function returns an error if there are non-displayed models in the session or if the input model is not displayed.

Note

- If the link of a TIM or ATB-enabled feature is broken, you cannot re-establish the link or update the part that is independent and has lost its association with the reference model.
- The geometry added or removed from the model before the update is added or removed from the TIM after the update.
- The geometry added or removed from the model before the update is added or removed from the TIM after the update.
- ATB incorrectly identifies the imported geometry as up-to-date based on the old reference file which is found before the updated reference file.

The function `ProATBRelink()` relinks a TIM to a source CAD model specified by the input argument master_model_path. This function relinks all those models or features that have lost their association or link with their master model. In order to relink a model, provide the name and location of the master model, using the master_model_path to which the specified model or feature is to be linked. If the master model with the same name is found, the Creo Parametric TIM model is linked to that master model.

30

Interface: Customized Plot Driver

This chapter describes the customized plot driver functions supported by Creo Parametric TOOLKIT .

Using the Plot Driver Functionality

The functions described in this section enable you to plot the two-dimensional entities, such as, lines, circles, arcs, and text that are used by Creo Parametric. These functions enable you to implement a customized plot format in Creo Parametric. You can do this by providing your own function for plotting each of the two-dimensional entities and then binding them to Creo Parametric so that it uses these functions to plot the contents of the object currently in session.

The following restrictions apply while using this functionality:

1. You cannot plot shaded models using these functions. This applies to plots of models generated from 3D model modes as well as shaded drawing views.
2. The contents of OLE objects embedded in drawings will not be included in the output.
3. Text contents provided to the implementation may include special Creo Parametric symbols. The Creo Parametric TOOLKIT application must transform this text into something that can be displayed correctly in the output format.
4. User defined plotter formats registered using the functions listed here will not be accessible from the Creo Parametric Print dialog box, or from the Creo Parametric TOOLKIT function `ProPrintExecute()`. In order to export the model with this custom plot driver, you must use `ProPlotdriverExecute()`.

Functions Introduced:

- **ProPlotdriverInterfaceCreate()**
- **ProPlotdriverInterfaceobjectsSet()**
- **ProPlotdriverArcPlot()**
- **ProPlotdriverArcfunctionSet()**
- **ProPlotdriverCirclePlot()**
- **ProPlotdriverCirclefunctionSet()**
- **ProPlotdriverLinePlot()**
- **ProPlotdriverLinefunctionSet()**
- **ProPlotdriverPolygonPlot()**
- **ProPlotdriverPolygonfunctionSet()**
- **ProPlotdriverPolylinePlot()**
- **ProPlotdriverPolylinefunctionSet()**
- **ProPlotdriverTextPlot()**

- **ProPlotdriverTextfunctionSet()**
- **ProPlotdriverExecute()**

The function `ProPlotdriverInterfaceCreate()` searches for a plot interface with a specified name. If the interface does not exist, then the function creates a new interface with this name. Each plot format that you define must have a unique name. This name is referenced in calls to other functions described in this section.

The function `ProPlotdriverInterfaceobjectsSet()` specifies the types of objects that are supported by a particular plot format. Specify a list of file extensions, such as, prt, drw, asm, and so on, that define the types of object to plot.

Use the `ProPlotdriver` functions to plot circles, lines, arcs, polygons, polylines, and text. These functions accept as input a call back function whose signature matches the call backs specified in the table below. The input arguments of the call back function describe the actual entity to be plotted.

The following table lists the names of the plot functions and their corresponding call backs. For example, use the function `ProPlotdriverLinefunctionSet()` to plot a line. The function accepts as input a call back function whose signature matches `ProPlotdriverLinePlot()`. The input arguments of the function `ProPlotdriverLinePlot()` define the line to be plotted.

Plot Function	Call Back Function
`ProPlotdriverArcfunctionSet()`	`ProPlotdriverArcPlot()`
`ProPlotdriverCirclefunctionSet()`	`ProPlotdriverCirclePlot()`
`ProPlotdriverLinefunctionSet()`	`ProPlotdriverLinePlot()`
`ProPlotdriverPolygonfunctionSet()`	`ProPlotdriverPolygonPlot()`
`ProPlotdriverPolylinefunctionSet()`	`ProPlotdriverPolylinePlot()`
`ProPlotdriverTextfunctionSet()`	`ProPlotdriverTextPlot()`

The function `ProPlotdriverExecute()` is used to invoke the user-defined plot on the current object.

Example 1: Sample Plot Driver Program

The sample code in `UgPlotUse.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_plot` shows how to use the customized plot driver functions.

31

Working with Multi-CAD Models Using Creo Unite

This chapter describes how to work with non-Creo parts and assemblies using Creo Unite.

Overview

Creo Unite enables you to open non-Creo parts and assemblies in Creo Parametric and other Creo applications, such as, Creo Simulate without creating separate Creo models. You can then assemble the part and assembly models that you opened as components of Creo assemblies to create multi-CAD assemblies of mixed content.

The non-Creo components of these heterogeneous assemblies retain their original names in Creo and continue to use their original source CAD applications as the design tool. They appear as foreign models and not as native Creo models.

You can open the part and assembly models of the following non-Creo file formats in Creo applications:

- CATIA V5 (`.CATPart`, `.CATProduct`)
- CATIA V5 CGR
- CATIA V4 (`.Model`)
- SolidWorks (`.sldasm`, `.sldprt`)
- NX (`.prt`)
- Autodesk Inventor (`.ipt`, `.iam`)
- Creo Elements/Direct (`.sdpc`, `.sdac`, `.sdcc`, and `.sdrc`)

  From Creo Parametric 4.0 F000 onward, the following Creo Elements/Direct files can be opened in Creo Parametric and other Creo applications, such as, Creo Simulate without creating separate Creo models:

  - Part content file (`.sdpc`)
  - Assembly content file (`.sdac`)

You can modify the non-Creo models in Creo applications, without altering the original design intent. For example, you can add annotations to the non-Creo models in a Creo application.

You can also make design changes to the non-Creo models in a multi-CAD assembly. Depending on the configuration options set in Creo, user confirmation may be required to apply the design changes. Refer to the Creo Parametric Data Exchange online help, for more information.

In applications where user confirmation cannot be obtained for design changes, for example, when Creo is running in batch mode, the appropriate functions such as, `ProFeatureWithoptionsCreate()`, return an error.

While working with a multi-CAD model, when you call the function `ProFeatureWithoptionsCreate()`, the output may vary depending on the value of the configuration option `confirm_on_edit_foreign_models`.

The default value of the configuration option `confirm_on_edit_foreign_models` is *yes*. The following scenarios are possible depending on the value of the configuration option `confirm_on_edit_foreign_models`:

- If the configuration option `confirm_on_edit_foreign_models` is set to *no*, the non-Creo model is modified without any notification.
- If the configuration option `confirm_on_edit_foreign_models` is set to *yes*, or the option is not defined in the configuration file, then in batch mode the application returns the error `PRO_TK_GENERAL_ERROR`.
- In some situations, you may need to provide input in the interactive mode with Creo. Refer to the Creo Parametric Data Exchange online help, for more information.

Support for File Names in Non-Creo Models

Creo applications support the original file names of non-Creo models. File name can contain a maximum of 80 characters and file paths can contain a maximum of 260 characters. The Creo Parametric TOOLKIT functions that work with multi-CAD models support file names with a maximum of 31 or 80 characters, depending on the type of function.

The functions that read information from Creo applications do not have any restriction on character length. These functions can read file names of any length. However, the functions that compute a result, or create or set features, item, properties, and so on, may have restrictions on character length. Refer to the Creo Parametric TOOLKIT header files, or the section Restrictions on Character Length for Multi-CAD Functions on page 758, for more information on functions that have restrictions on character length.

Character Support for File Names in Non-Creo Models

The following special characters are supported for file names in non-Creo models:

- % (percent)
- ^ (caret)
- & (ampersand)
- + (plus)
- = (equal)
- ' (apostrophe)
- ` (grave accent)

- , (comma)
- ! (exclamation mark)
- $ (dollar sign)
- @ (at sign)
- ; (semicolon)
- # (hash)
- - (dash, hyphen)
- ~ (tilde)
- ( ) (round brackets)
- [ ] (square brackets)
- { } (curly brackets)
- . (period)
- \ (backslash)
- / (forward slash)
- “ (quotation marks)
- (space)

Note

Do not use space as the first character in file names for models.

Working with Multi-CAD Models in Creo Parametric TOOLKIT

Most of the Creo Parametric TOOLKIT functions support multi-CAD assemblies. The functions, which do not support assemblies with mixed content, return the error `PRO_TK_UNSUPPORTED`.

You can perform basic operations, such as, locating and retrieving non-Creo models from Windchill and opening them in Creo applications. However, you must use Windchill Workgroup Manager to initially check in the non-Creo models to Windchill. After checking out models from Windchill, you can work on them in Creo applications and then check in the models to Windchill.

Working with Notifications

Notifications allow the Creo Parametric TOOLKIT application to detect certain types of events in Creo Parametric. You can call functions before or after such events.

Notifications to all of the file management operations in Creo Parametric, such as save, retrieve, copy, rename, and so on, are supported for Creo assemblies with non-Creo components.

The following types of file management notifications are supported for multi-CAD assemblies:

- File Management Events—Notifications that are called after successful file management operations in Creo Parametric.
  - Pre-file Management Events—Your callback function is called before the file management event. The functions are called only for models that are the explicit objects of the file management operation.
  - Pre-All File Management Events—Your callback function is called before all file management events on models, even if those models were not explicitly specified by the user.
  - Post-file Management Events—Your callback function is called after the file management operation. The functions are called only for models that are the explicit objects of the file management operation.
  - Post All File Management Events—Your callback function is called after all file management events on models, even if those models were not explicitly specified by the user.
- File Management Failure Events—Notifications that are called after the file management operations in Creo Parametric fail.

The following events and callback functions are supported for multi-CAD assemblies:

New Event	New Signature
Pre-file Management Events	
PRO_MODEL_RETRIEVE_PRE	(*ProModelRetrievePreAction)
PRO_MODEL_SAVE_PRE	(*ProModelSavePreAction)
PRO_MODEL_COPY_PRE	(*ProModelCopyPreAction)
PRO_MODEL_RENAME_PRE	(*ProModelRenamePreAction)
Pre-All File Management Events	
PRO_MODEL_SAVE_PRE_ALL	(*ProModelSavePreAllAction)
Post File Management Events	
PRO_MODEL_COPY_POST	(*ProModelCopyPostAction)
PRO_MODEL_RENAME_POST	(*ProModelRenamePostAction)
PRO_MODEL_ERASE_POST	(*ProModelErasePostAction)
PRO_MODEL_RETRIEVE_POST	(*ProModelRetrievePostAction)
PRO_MODEL_SAVE_POST	(*ProModelSavePostAction)
Post All File Management Events	

New Event	New Signature
PRO_MODEL_SAVE_POST_ALL	(*ProModelSavePostAllAction)
PRO_MODEL_ERASE_POST_ALL	(*ProModelErasePostAllAction)
PRO_MODEL_RETRIEVE_POST_ALL	(*ProModelRetrievePostAllAction)
PRO_MODEL_COPY_POST_ALL	(*ProModelCopyPostAllAction)
PRO_MODEL_RENAME_POST_ALL	(*ProModelRenamePostAllAction)
File Management Failed Events	
PRO_MODEL_DBMS_FAILURE	(*ProModelDbmsFailureAction)

Working with Basic Graphics

You can create windows, which contain the specified multi-CAD assemblies. The functions `ProObjectwindowMdlnameCreate()`, `ProAccessorywindowWithTreeMdlnameCreate()`, and `ProBarewindowMdlnameCreate()` are used to create windows. To use these functions with non-Creo components, pass one of the following values as the input argument `object_type`, depending on the type of non-Creo model:

- `PRO_CATIA_PART`
- `PRO_CATIA_PRODUCT`
- `PRO_CATIA_CGR`
- `PRO_CATIA_MODEL`
- `PRO_UG`
- `PRO_SW_PART`
- `PRO_SW_ASSEM`
- `PRO_CC_ASSEMBLY`
- `PRO_CC_PART`

Working with Simplified Representations

You can create simplified representations, that is, master, geometric, graphics, and user-defined representations, for multi-CAD assemblies. You can also retrieve these representations in the session using the Creo Parametric TOOLKIT functions. You can edit a simplified representation created in a Creo application for a non-Creo assembly.

Note

The function `ProSolidSimprepVisit()` visits only user-defined representations. Multi-CAD assemblies can have user-defined representations, but the non-Creo parts cannot have user-defined representations. Therefore, the function `ProSolidSimprepVisit()` returns `PRO_TK_E_NOT_FOUND` error for non-Creo parts.

Working with Constraints

You can retrieve the constraints of non-Creo models in multi-CAD assemblies. For retrieving the constraints, you must redefine the non-Creo models using the **Edit Definition** command in the Creo Parametric user interface. After redefinition, retrieve the constraints using the Creo Parametric TOOLKIT functions.

Working with User-Defined Features (UDF)

You can insert user-defined features created in Creo applications in a multi-CAD assembly, only if it does not alter the structure of the non-Creo models. You cannot create UDFs in a non-Creo model.

Functions that Support Multi-CAD Assemblies

The following functions support working with Creo assemblies with mixed content:

- **ProIntfimportModelWithOptionsMdlnameCreate()**
- **ProMdlDependenciesDataList()**
- **ProMdlMdlnameGet()**
- **ProMdlfileMdlnameCopy()**
- **ProMdlnameBackup()**
- **ProMdlnameCopy()**
- **ProMdlnameInit()**
- **ProMdlnameRename()**
- **ProMdlnameRetrieve()**
- **ProMdlDirectoryPathGet()**
- **ProMdlExtensionGet()**
- **ProMdlDeclaredDataList()**

- **ProMfgMdlCreate()**
- **ProReferenceOriginalownerMdlnameGet()**
- **ProReferenceOwnerMdlnameGet()**
- **ProSolidShrinkwrapMdlnameCreate()**
- **ProAssemblySimprepMdlnameRetrieve()**
- **ProSimprepMdlnameRetrieve()**
- **ProAsmSkeletonMdlnameCreate()**
- **ProSolidMdlnameCreate()**
- **ProSolidMdlnameInit()**
- **ProUdfmdlMdlnamesAlloc()**
- **ProFileMdlnameParse()**
- **ProOutputFileMdlnameWrite()**
- **ProFileMdlnameOpen()**
- **ProFileMdlnameSave()**
- **ProIntfSliceFileWithOptionsMdlnameExport()**
- **Pro2dImportMdlnameCreate()**
- **ProPathMdlnameCreate()**
- **ProProductviewFormattedMdlnameExport()**
- **ProObjectwindowMdlnameCreate()**
- **ProAccessorywindowWithTreeMdlnameCreate()**
- **ProBarewindowMdlnameCreate()**
- **ProATBMdlnameVerify()**
- **ProAsmcompMdlMdlnameGet()**
- **ProAsmcompMdlnameCreateCopy()**
- **ProCavitylayoutLeaderMdlnameSet()**
- **ProCavitylayoutModelMdlnamesGet()**
- **ProCavitylayoutModelMdlnamesSet()**
- **ProMdlFiletypeLoad()**
- **ProMdlRepresentationFiletypeLoad()**

For some of these functions, you may have to specify the type of the non-Creo CAD model using the data type `ProMdlfileType`. The valid values of `ProMdlfileType` are:

- `PRO_CATIA_PART`
- `PRO_CATIA_PRODUCT`
- `PRO_CATIA_CGR`
- `PRO_CATIA_MODEL`

- PRO_UG
- PRO_SW_PART
- PRO_SW_ASSEM
- PRO_INVENTOR_PART
- PRO_INVENTOR_ASSEM

Superseded Functions

The following table lists the functions that have been superseded and the corresponding new functions to support multi-CAD assemblies:

Superseded Function	New Function
ProIntfimportModelWithOptionsCreate()	ProIntfimportModelWithOptionsMdlnameCreate()
ProMdlDependenciesList()	ProMdlDependenciesDataList()
ProMdlNameGet()	ProMdlMdlnameGet()
ProMdlfileCopy()	ProMdlfileMdlnameCopy()
ProMdlBackup()	ProMdlnameBackup()
ProMdlCopy()	ProMdlnameCopy()
ProMdlInit()	ProMdlnameInit()
ProMdlRename()	ProMdlnameRename()
ProMdlRetrieve()	ProMdlnameRetrieve()
ProMdlDataGet()	ProMdlOriginGet() ProMdlMdlnameGet() ProMdlExtensionGet() ProFilenameParse()
ProMdlDeclaredList()	ProMdlDeclaredDataList()
ProMfgCreate()	ProMfgMdlCreate()
ProReferenceOriginalownernameGet()	ProReferenceOriginalownerMdlnameGet()
ProReferenceOwnernameGet()	ProReferenceOwnerMdlnameGet()
ProSolidShrinkwrapCreate()	ProSolidShrinkwrapMdlnameCreate()
ProAssemblySimprepRetrieve()	ProAssemblySimprepMdlnameRetrieve()
ProBoundaryBoxSimprepRetrieve()	ProSimprepMdlnameRetrieve()
ProDefaultEnvelopeSimprepRetrieve()	ProSimprepMdlnameRetrieve()
ProGraphicsSimprepRetrieve()	ProSimprepMdlnameRetrieve()
	ProSimprepMdlnameRetrieve()

Superseded Function	New Function
ProLightweightGraphicsSimprepRetrieve()	
ProPartSimprepRetrieve()	ProSimprepMdlnameRetrieve()
ProSymbSimprepRetrieve()	ProSimprepMdlnameRetrieve()
ProGeomSimprepRetrieve()	ProSimprepMdlnameRetrieve()
ProAsmSkeletonCreate()	ProAsmSkeletonMdlnameCreate()
ProSolidCreate()	ProSolidMdlnameCreate()
ProSolidInit()	ProSolidMdlnameInit()
ProUdfmdlNamesAlloc()	ProUdfmdlMdlnamesAlloc()
ProFilenameParse()	ProFileMdlnameParse()
ProOutputFileWrite()	ProOutputFileMdlnameWrite()
ProFileOpen()	ProFileMdlnameOpen() ProFileMdlfiletypeOpen()
ProFileSave()	ProFileMdlnameSave() ProFileMdlfiletypeSave()
ProIntfSliceFileWithOptionsExport()	ProIntfSliceFileWithOptionsMdlnameExport()
Pro2dImportCreate()	Pro2dImportMdlnameCreate()
ProPathCreate()	ProPathMdlnameCreate()
ProProductviewFormattedExport()	ProProductviewFormattedMdlnameExport()
ProObjectwindowCreate()	ProObjectwindowMdlnameCreate()
ProAccessorywindowWithTreeCreate()	ProAccessorywindowWithTreeMdlnameCreate()
ProBarewindowCreate()	ProBarewindowMdlnameCreate()
ProATBVerify()	ProATBMdlnameVerify()
ProAsmcompMdlNameGet()	ProAsmcompMdlMdlnameGet()
ProAsmcompCreateCopy()	ProAsmcompMdlnameCreateCopy()
ProCavitylayoutLeaderSet()	ProCavitylayoutLeaderMdlnameSet()
ProCavitylayoutModelnamesGet()	ProCavitylayoutModelMdlnamesGet()
ProCavitylayoutModelnamesSet()	ProCavitylayoutModelMdlnamesSet()
ProMdlLoad()	ProMdlFiletypeLoad()
ProMdlRepresentationLoad()	ProMdlRepresentationFiletypeLoad()
(*ProMdlSavePreAction)	(*ProModelSavePreAction)
(*ProMdlCopyPreAction)	(*ProModelCopyPreAction)
(*ProMdlRenamePreAction)	(*ProModelRenamePreAction)

Superseded Function	New Function
(*ProMdlRetrievePreAction)	(*ProModelRetrievePreAction)
(*ProMdlSavePostAction)	(*ProModelSavePostAction)
(*ProMdlCopyPostAction)	(*ProModelCopyPostAction)
(*ProMdlRenamePostAction)	(*ProModelRenamePostAction)
(*ProMdlErasePostAction)	(*ProModelErasePostAction)
(*ProMdlRetrievePostAction)	(*ProModelRetrievePostAction)
(*ProMdlSavePostAllAction)	(*ProModelSavePostAllAction)
(*ProMdlCopyPostAllAction)	(*ProModelCopyPostAllAction)
(*ProMdlErasePostAllAction)	(*ProModelErasePostAllAction)
(*ProMdlRetrievePostAllAction)	(*ProModelRetrievePostAllAction)
(*ProMdlDbmsFailureAction)	(*ProModelDbmsFailureAction)

Restrictions on Character Length for Multi-CAD Functions

This section describes the restriction on character lengths for Creo Parametric TOOLKIT functions that support Creo assemblies with mixed content.

The following table lists the maximum number of characters supported by these functions:

Function Name	Character Length Supported	Additional Comment
ProSolidShrinkwrapMdl-nameCreate()	31 characters	
ProSolidMdlnameCreate ()	31 characters	
ProSolidMdlnameInit()	80 characters	
ProAssemblySim-prepMdlnameRetrieve()	80 characters	
ProSimprepMdlnameRe-trieve()	80 characters	
ProAsmSkeletonMdlna-meCreate()	31 characters	
ProOutputFileMdlname-Write()	31 characters	
ProFileMdlnameParse()	80 characters	
ProFileMdlnameOpen()	80 characters	
ProFileMdlnameSave()	80 characters	

Function Name	Character Length Supported	Additional Comment
ProFileMdlfiletypeOpen()	80 characters	
ProFileMdlfiletypeSave()	80 characters	
ProMdlDependenciesDataList()	80 characters	
ProMdlfileMdlnameCopy()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProMdlfileCopy()`. It returns an error for model names longer than 31 characters.
ProMdlnameBackup()	80 characters	
ProMdlnameCopy()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProMdlCopy()`. It returns an error for model names longer than 31 characters. **Note** This function is not supported for non-Creo models and Creo assemblies with mixed content.
ProMdlnameInit()	80 characters	
ProMdlnameRetrieve()	80 characters	
ProMdlnameRename()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProMdlRename()`. It returns an error for model names longer than 31 characters.

Function Name	Character Length Supported	Additional Comment
		Note This function is not supported for non-Creo models and Creo assemblies with mixed content.
ProMdlDeclaredDataList()	80 characters	
ProMfgMdlCreate()	31 characters	
ProIntfimportModelWithOptionsMdlnameCreate()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProIntfimportModelWithOptionsCreate()`. It returns an error for model names longer than 31 characters.
ProIntfSliceFileWithOptionsMdlnameExport()	31 characters	
Pro2dImportMdlnameCreate()	31 characters	
ProProductviewFormattedMdlnameExport()	31 characters	
ProObjectwindowMdlnameCreate()	80 characters	
ProAccessorywindowWithTreeMdlnameCreate()	80 characters	
ProBarewindowMdlnameCreate()	80 characters	
ProATBMdlnameVerify()	80 characters	
ProAsmcompMdlnameCreateCopy()	31 characters	
ProCavitylayoutLeaderMdlnameSet()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProCavitylayout`

Function Name	Character Length Supported	Additional Comment
		`LeaderSet()`. It returns an error for model names longer than 31 characters.
ProCavitylayoutModelMdlnamesSet()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProCavitylayoutModelnamesSet()`. It returns an error for model names longer than 31 characters.
ProUdfmdlMdlnamesAlloc()	31 characters	In Creo Parametric 3.0, this API is functionally similar to the superseded function `ProUdfmdlNamesAlloc()`. It returns an error for model names longer than 31 characters. **Note** This function is not supported for non-Creo models and Creo assemblies with mixed content.
ProPathMdlnameCreate()		No restriction on character length
ProCavitylayoutModelMdlnamesGet()		No restriction on character length
ProMdlDirectoryPathGet()		No restriction on character length
ProMdlFiletypeLoad()		No restriction on character length
ProMdlRepresentationFiletypeLoad()		No restriction on character length
ProMdlFileTypeGet()		No restriction on character length
ProMdlExtensionGet()		No restriction on character length

Function Name	Character Length Supported	Additional Comment
ProReferenceOriginalowner MdlnameGet()		No restriction on character length
ProReferenceOwnerMdlnameGet()		No restriction on character length
ProMdlMdlnameGet()		No restriction on character length
ProAsmcompMdlMdlnameGet()		No restriction on character length

Functional Areas in Creo that do not Support Multi-CAD Assemblies

You cannot work with multi-CAD assemblies in the following functional areas:

- Family tables
- Flexible components
- Rename, delete, and copy operations for non-Creo models in a multi-CAD assembly

> **Note**
> When you save a copy, the non-Creo models are saved as Creo models.

- Add, delete, replace, reorder, and supress components in a non-Creo assembly
- Move components to a new subassembly in a non-Creo assembly
- Creation of external references. However, you can create external references, only if you have set the external reference option in Creo Parametric user interface.

Sample Applications for Multi-CAD Assemblies

The sample applications provided with the Creo Parametric TOOLKIT 3.0 installation use the functions that support multi-CAD assemblies. The sample applications are available at `<creo_loadpoint>\<datecode>\Common Files\protoolkit\protk_appls`. See the sample examples for more information on how to migrate your existing applications to support multi-CAD assemblies using the Creo Parametric TOOLKIT 3.0 functions.

Migrating Applications Using Tools

PTC recommends migrating your applications to the new interfaces, which would support multi-CAD models in future releases.

A mapping table `mcm_map.txt`, provided with the Creo Parametric TOOLKIT 3.0 installation, lists all the functions that support multi-CAD assemblies and also lists the deprecated functions.

You can migrate the applications using the perl script `mark_deprecated.pl`, which is available at the location `<creo_loadpoint>\<datecode>\Common Files\protoolkit\scripts`. Refer to the section Tools Available for Migration on page 2085 in the appendix Migrating Older Applications on page 2084, for more information on the script.

32

Element Trees: Principles of Feature Creation

This chapter describes the basic principles of programmatic feature creation that are applicable to all types of feature that can be created in the current version of Creo Parametric TOOLKIT. This chapter also describes how to extract the internal description of features of those feature types in the Creo Parametric TOOLKIT database, and how to modify them.

Overview of Feature Creation

This section provides references to additional material on feature creation and an overview of creating features.

References to Feature Creation Data

The creation of specific feature types is dealt with in more detail in the following chapters:

- Element Trees: Datum Features on page 804
- Element Trees: Sketched Features on page 1004
- Production Applications: Manufacturing on page 1439

This chapter defines and describes the following Creo Parametric TOOLKIT objects:

- `ProElement`
- `ProElempath`
- `ProValue`

For a definition of the `ProFeature` object, see the chapter Core: Features on page 131.

Feature Creation

There are many kinds of feature in Creo Parametric and each one can contain a large and varied amount of information. All this information must be complete and consistent before a feature can be used in regeneration and give rise to geometry. This raises several problems for programmatic creation of features through an API.

It is necessary to build all the information needed by a feature into a data structure before passing that whole structure to Creo Parametric. However, the object-oriented style of Creo Parametric TOOLKIT requires that such a data structure is not directly visible to the application. Therefore, Creo Parametric TOOLKIT defines this structure as a workspace object that can be allocated and filled using special functions for that purpose, but that is not part of the Creo Parametric database.

There are three steps in creating a feature in Creo Parametric:

1. Allocate the workspace structure.
2. Fill the workspace structure.
3. Pass the workspace structure to Creo Parametric to create the feature.

Note

Creating sketched features requires a few more steps. For detailed information, see the chapter Element Trees: Sketched Features on page 1004.

The procedure described above allows a feature of arbitrary complexity to be built up in a sequence of manageable steps, with the maximum of error checking along the way.

Although it is not yet possible to create all feature types using Creo Parametric TOOLKIT, the workspace structure must be capable of defining any feature type so the range of features can be extended without affecting the techniques already in use. For this reason, the workspace structure for feature creation takes the form of a tree of data elements. This has the advantage of being simple for simple features, yet is flexible enough to provide for all possible feature types without introducing new principles.

The root and branch points in the tree are called “elements,” and the complete tree is called the “feature element tree.” Each element is modeled by the object `ProElement`, which is a pointer to an opaque structure.

The feature element tree contains all the information required to define the feature. This includes the following:

- All options and attributes, such as the material side and depth type for an extrusion or slot, placement method for a hole, and so on.
- All references to existing geometry items, such as placement references, “up to” surfaces, sketching planes, and so on.
- References to Sketcher models used for sections in the feature.
- All dimension values.

Note

Because Creo Parametric TOOLKIT is the same toolkit used to build Creo Parametric, improvements to Creo Parametric may require the definition of the element tree to be altered for some features. PTC will make every effort to maintain upward compatibility. However, there may be cases where the old application will not run with the new version of Creo Parametric, unless you rewrite the application's code to conform to the new definition of the feature tree.

Note that although the values of dimensions used by the feature are in the element tree, there are no descriptions of, or references to, the dimension objects themselves. The only exception is as follows: for an element tree for a feature already in the Creo Parametric database, you can ask the identifier of the dimension used for a particular element using the function `ProFeatureElemDimensionIdGet()`. This is explained in detail in the section Feature Inquiry on page 785. For more general functions that access dimensions, see the chapter Core: Relations on page 204.

Each element in the tree is assigned an element ID, which is really a description of the role it is playing in this feature—the kind of information it is supplying. It is called an element ID because no two elements at the same level in the tree will have the same identifier, unless they belong to an array element, so the element ID also acts as a unique identifier.

The possible element roles are values in an enumerated type called `ProElemId`, declared in `ProElemId.h`. Example values are as follows:

- `PRO_E_FEATURE_TYPE`
- `PRO_E_FEATURE_FORM`
- `PRO_E_EXT_DEPTH`
- `PRO_E_THICKNESS`
- `PRO_E_4AXIS_PLANE`

There are four different element types:

- Single-valued
- Multivalued
- Compound
- Array

A single-valued element can contain various types of value. The simplest is an integer used to define, for example, the type of the feature, or one of the option choices, such as the material side for a thin protrusion. The value can be a wide string (for example, the name of the feature), or a double (for example, the depth of a blind extrusion). If the element defines a reference to an existing geometry item in the solid, its value contains an entire `ProSelection` object so it can refer to anything in an entire assembly.

A multivalued element contains several values of one of these types. Multivalued elements occur at the lowest level of the element tree—the “leaves.” An example is the element with the identifier `PRO_E_FIXT_COMPONENTS` in a Fixture Setup feature in Creo NC. That element specifies the components in the assembly that belong to the fixture; in general, there can be any number of such components, so the element contains several component identifiers.

A compound element is one that acts as a branch point in the tree. It does not have a value of its own, but acts as a container for elements further down in the hierarchy.

An array element is also a branch point, but one that contains many child elements of the same element ID. An example of this is the `PRO_E_DTMPLN_CONSTRAINTS` element in a datum plane feature, which contains an array of elements of type `PRO_E_DTMPLN_CONSTRAINT` (note the singular), each of which is a compound element whose contents describe one of the constraints that determine the position of the datum plane.

The feature element tree enables you to build a complex feature in stages, with only a small set of functions. However, the form of the tree required for a particular feature needs to be clearly defined so you know exactly what elements and values to add, and so Creo Parametric TOOLKIT can check for errors each time you add a new element to the tree.

Creo Parametric TOOLKIT documents the necessary contents of the element tree for each type of feature that can be created programmatically. It does this through two types of description:

- Feature element tree
- Feature element table

The feature element tree defines the structure of the tree, specifying the element ID (or role) for the elements at all levels in the tree, and which elements are optional.

The feature element table defines the following for each of the element IDs in the tree:

- A description of its role in the feature
- The value type it has (that is, whether it is single value or compound; or an array of integer, double, or string)
- The range of values valid for it in this context

Each type of feature that can be created using Creo Parametric TOOLKIT has its own header file that contains the feature element tree and table, in the form of code comments. The header files for the feature types that can be created in the current version are as follows:

Header File	Feature Type
`ProAnalysis.h`	External Analysis
`ProAsmcomp.h`	Assembly component, mechanism connection
`ProChamfer.h`	Chamfer or corner chamfer
`ProDataShareFeat.h`	General merge (merge, cutout, inheritance), copy geom., publish geom., shrinkwrap
`ProDraft.h`	Draft
`ProDtmAxis.h`	Datum axis
`ProDtmCrv.h`	Datum curve

Header File	Feature Type
ProDtmCsys.h	Datum coordinate system
ProDtmPln.h	Datum plane
ProDtmPnt.h	Datum point
ProExtrude.h	Extruded protrusion, cut, surface, surface trim, thin, sheetmetal wall, or sheetmetal cut
ProFixture.h	Fixture (for Creo NC)
ProFlatSrf.h	Fill surface or sheetmetal flat wall
ProForeignCurve.h	Foreign datum curve
ProHole.h	Hole
ProMerge.h	Merge
ProMirror.h	Mirror (geometry only)
ProMove.h	Move and copy (geometry only)
ProMfgoper.h	Manufacturing operation
ProNcseq.h	Manufacturing Creo NC sequence
ProProcstep.h	Process step
ProReplace.h	Surface replacement feature
ProRevolve.h	Revolved protrusion, cut, surface, surface trim or thin
ProRib.h	Rib
ProRound.h	Round
ProShell.h	Shell
ProSmtFlangeWall.h	Sheetmetal flange wall
ProSmtFlatWall.h	Sheetmetal flat wall
ProSolidify.h	Solidify
ProStdSection.h	Standard section
ProSweep.h	Simple swept protrusion, cut
ProThicken.h	Thicken
ProTrim.h	Trim
ProWcell.h	Manufacturing workcell

The feature element tree for the Fixture Setup feature defined in the header file `ProFixture.h` is deprecated. Use the element tree defined in the header file `ProMfgFeatFixture.h` instead. For more information, please refer to the section Manufacturing Holemaking Step on page 1578.

The first two elements are common to all features. The root of a feature tree is always a compound element with the identifier `PRO_E_FEATURE_TREE`. The first element within the root always specifies the feature type; it is a single-valued element with the element ID `PRO_E_FEAT_TYPE`, whose value is one of the integers in the list of feature types in `ProFeatType.h`. In this case, the element table shows that the value must be `PRO_FEAT_FIXSETUP`.

The next element in a fixture setup gives the name of the feature; its element ID is `PRO_E_FEAT_NAME`, and it contains a single wide string. The element tree shows that this is optional.

The PRO_E_FIXT_COMPONENTS is a multivalued element, with the value type integer, which contains the identifiers of the assembly components that belong to the fixture.

The last element in a fixture setup is PRO_E_SETUP_TIME, which contains a double.

As you build the elements into the workspace element tree, Creo Parametric TOOLKIT checks the correctness of their types against the structure described by the element tree and table in the corresponding header file. This makes it easy to diagnose errors when you are creating features. The geometrical correctness is checked only when you try to create the feature in the Creo Parametric database.

The following sections of this chapter describe the functions used to build an element tree and create a feature. The sections are as follows:

- Feature Element Values on page 770—Introduces the object ProValue, used to represent the value of a feature element.
- Feature Element Paths on page 772—Introduces the object ProElempath, used to describe the location of an element in an element tree.
- Feature Elements on page 774—Introduces the ProElement functions used to build and analyze an element tree.
- Feature Element Diagnostics on page 778—Introduces the ProElementDiagnostics functions used to obtain the diagnostics for a feature element.
- Calling ProFeatureCreate() on page 779—Describes the ProFeatureCreate() function in detail.
- Example of Complete Feature Creation on page 781—Shows how to use functions from the other sections to perform all the steps needed to create a feature.

Feature Element Values

Functions introduced:

- **ProValueAlloc()**
- **ProValueDataGet()**
- **ProValueDataSet()**
- **ProValueFree()**
- **ProWstringArrayToValueArray()**
- **ProValueArrayToWstringArray()**
- **ProValuedataTransformGet()**
- **ProValuedataTransformSet()**

The object ProValue is an opaque workspace handle used to contain the value of a feature element. It is the output of the functions ProElementValueGet() and ProElementValuesGet(), which read the values of a feature element, and is the input to ProElementValueSet() and ProElementValuesSet().

You can access the contents of a ProValue object by translating it into an object of type ProValueData, which is declared as a visible data structure. The declaration is as follows:

```
typedef struct pro_value_data
{
  ProValueDataType   type;
  union
  {
    int             i;    /* integer */
    double          d;    /* double */
    void           *p;    /* pointer or reference */
    char           *s;    /* string */
    wchar_t        *w;    /* wchar_t */
    ProSelection    r;    /* selection */
  } v;
} ProValueData;
  typedef enum pro_value_data_type
  {
    PRO_VALUE_TYPE_INT           = 1,
    PRO_VALUE_TYPE_DOUBLE,
    PRO_VALUE_TYPE_POINTER,
    PRO_VALUE_TYPE_STRING,
    PRO_VALUE_TYPE_WSTRING,
    PRO_VALUE_TYPE_SELECTION
  } ProValueDataType;
```

ProValueData is simply a holder for data values of many different types.

Note

From Pro/ENGINEER Wildfire 2.0 onwards, elements with multiple values, for example, PRO_E_FIXT_COMPONENTS, are deprecated. In subsequent releases, these elements will be superseded by reference elements or single-value, type-specific elements. Use the function ProElementValuetypeGet() to determine the type of the element.

Note

To access reference elements use the functions `ProElementReferencesGet()` or `ProElementReferencesSet()`. To access single-value, type-specific elements, use the functions `ProElement<type>Get()` or `ProElement<type>Set()`, such as, `ProElementDoubleGet()`, `ProElementIntegerGet()` and so on.

The functions in this section access the contents of a `ProValue` through the `ProValueData` object.

The function `ProValueDataGet()` provides the `ProValueData` object for the specified `ProValue` object.

The function `ProValueAlloc()` allocates a new `ProValue` in memory, as the first step towards setting the value of a feature element.

The function `ProValueDataSet()` sets the value of a `ProValue` object using the contents of a `ProValueData` structure.

The function `ProValueFree()` frees a `ProValue` object in memory.

The function `ProWstringArrayToValueArray()` provides a convenient way to allocate and fill an array of `ProValue` structures that all contain wide string values.

The function `ProValueArrayToWstringArray()` performs the reverse translation, allocating and filling an array of wide strings. In both cases, the output array is an expandable array, so you should release the memory using `ProArrayFree()`.

The transform member of the union ProValueData is declared as double**. It must be passed a double[][] (a `ProMatrix` structure). The utility functions `ProValuedataTransformGet()` and `ProValuedataTransformSet()` specify how to assign the `ProValueData` in order to access the matrix correctly.

Feature Element Paths

Functions introduced:

- **ProElempathAlloc()**
- **ProElempathFree()**
- **ProElempathDataSet()**
- **ProElempathDataGet()**
- **ProElempathCopy()**

- **ProElempathCompare()**
- **ProElempathSizeGet()**

An element path is used to describe the location of an element in an element tree. It is used by some of the `ProElement` functions as a convenient way to refer to elements already in a tree.

The object `ProElempath` is declared as an opaque pointer. It contains a description of the path from the root of the tree down to the element referred to. At most levels in the tree hierarchy, the relevant path member is the element ID of the element (which is unique at that level). When the path steps from an array element to one of its member arrays, the element path instead contains the array index of that element.

To be able to set the value of a `ProElempath`, Creo Parametric TOOLKIT provides a structure called `ProElempathItem` that can describe an element ID, or the index into an array element. An array of `ProElempathItem` structures is therefore a visible equivalent to the opaque contents of `ProElempath`.

The declaration of `ProElempathItem` is as follows:

```
typedef struct path
{
  ProElempathItemtype type;
  union
  {
    int elem_id;
    int elem_index;
  } path_item;
} ProElempathItem;
  typedef enum
  {
    PRO_ELEM_PATH_ITEM_TYPE_ID,
    PRO_ELEM_PATH_ITEM_TYPE_INDEX
  } ProElempathItemtype;
```

The object `ProElempath`, the structure `ProElempathItem`, and all the functions in this section are declared in the header file `ProElempath.h`.

The function `ProElempathAlloc()` allocates a new empty `ProElempath` object, whereas `ProElempathFree()` frees a `ProElempath`.

The function `ProElempathDataSet()` enables you to set the contents of a `ProElempath` by copying from an array of `ProElempathItem` structures.

The function `ProElempathDataGet()` reads the contents of a `ProElempath` into an array of `ProElempathItem` structures. The array is an expandable array that must be allocated by a call to `ProArrayAlloc()` before you call the function.

The function `ProElempathCopy()` copies the contents of one `ProElempath` object into another. The output object is allocated by the function.

The function `ProElempathCompare()` tells you whether two `ProElempath` objects refer to the same element.

The function `ProElempathSizeGet()` tells you the length of the element path contained in a `ProElempath` object.

Feature Elements

Functions introduced:

- **ProElementAlloc()**
- **ProElementFree()**
- **ProElementIdGet()**
- **ProElementIdSet()**
- **ProElemIdStringGet()**
- **ProElemtreeElementGet()**
- **ProElemtreeElementAdd()**
- **ProElemtreeElementRemove()**
- **ProElementIsMultival()**
- **ProElementIsCompound()**
- **ProElementIsArray()**
- **ProElementChildrenGet()**
- **ProElementChildrenSet()**
- **ProElementArraySet()**
- **ProElementArrayGet()**
- **ProElementArrayCount()**
- **ProElemtreeElementVisit()**

The function `ProElementAlloc()` allocates a new `ProElement` object with a specified element ID. The function `ProElementFree()` frees a `ProElement`.

The function `ProElementIdGet()` outputs the element ID of a specified `ProElement`. The function `ProElementIdSet()` enables you to set the element ID of a specified `ProElement`.

The function `ProElemIdStringGet()` returns the string representation of the specified element ID.

The function `ProElemtreeElementGet()` enables you to read a specified element in a tree. The inputs are the root of the tree, specified as a `ProElement` object, and the path to the element, specified by a `ProElempath`. The output is a `ProElement` object.

The function `ProElemtreeElementAdd()` adds a new element to the specified element tree. The inputs are the `ProElement` for the tree root, the `ProElempath` to the new element, and the `ProElement` for the new element.

The function `ProElemtreeElementRemove()` removes an element from a specified tree and path. It outputs a `ProElement` for the element removed.

The functions `ProElementIsMultival()`, `ProElementIsCompound()`, and `ProElementIsArray()` tell you the type of a specified element in a tree. See the section Overview of Feature Creation on page 765 for an explanation of the types.

The function `ProElementChildrenGet()` provides an expandable array of `ProElement` objects for the children of the specified compound element in a tree. The array must be allocated using `ProArrayAlloc()` before you call this function. The function `ProElementChildrenSet()` adds a set of elements, specified by an expandable array of `ProElement` objects, as the children of the specified element in a tree.

The function `ProElementArraySet()` adds an expandable array of `ProElement` objects as the members of a specified array element in an element tree.

The function `ProElementArrayGet()` fills an expandable `ProElement` array with the members of an array element in an element tree. The function `ProElementArrayCount()` tells you how many members are in an array element in the specified element tree.

The function `ProElemtreeElementVisit()` visits the elements that are members of the specified array element in an element tree.

Access to ProElement Data

In earlier releases, Pro/TOOLKIT recommended using functions that access the element value(s) as `ProValue` objects. These functions are maintained for compatibility. PTC recommends using the functions in this section to provide greater flexibility for all new development related to any Pro/ENGINEER feature type. Use of these new functions is required for features first supported in Pro/ENGINEER Wildfire 2.0 and later.

Functions introduced:

- **ProElementValuetypeGet()**
- **ProElementReferenceGet()**
- **ProElementReferenceSet()**
- **ProElementReferencesGet()**
- **ProElementReferencesSet()**
- **ProElementIntegerGet()**

- **ProElementIntegerSet()**
- **ProElementDoubleGet()**
- **ProElementDoubleSet()**
- **ProElementDecimalsGet()**
- **ProElementDecimalsSet()**
- **ProElementwstroptsAlloc()**
- **ProElementwstroptsExpressionsSet()**
- **ProElementwstroptsSignoptionsSet()**
- **ProElementwstroptsFree()**
- **ProElementWstringGet()**
- **ProElementWstringSet()**
- **ProElementStringGet()**
- **ProElementStringSet()**
- **ProElementSpecialvalueGet()**
- **ProElementSpecialvalueSet()**
- **ProElementBooleanGet()**
- **ProElementBooleanSet()**
- **ProElementTransformGet()**
- **ProElementTransformSet()**

These functions are the preferred method of accessing element values information over `ProElementValueGet()` or `ProElementValueSet()` and `ProValueDataGet()` or `ProValueDataSet()`.

The function `ProElementValuetypeGet()` returns the nominal value type for the element.

The functions `ProElementReferenceGet()` and `ProElementReferenceSet()` returns and sets a single reference value for the element.

The function `ProElementReferencesGet()` returns an array of reference values for the element. The function outputs a reference array, which is a `ProArray`. Free this output using `ProReferencearrayFree()`.

The function `ProElementReferencesSet()` sets the multiple reference values for the element.

The function `ProElementIntegerGet()` returns an integer value representation for the element. The function `ProElementIntegerSet()` sets the integer value for the element.

The function `ProElementDoubleGet()` returns a double value representation for the element. The function `ProElementDoubleSet()` sets the double value for the element.

The function `ProElementDecimalsGet()` obtains the number of decimal places to be used for the double value of an element in the feature.

The function `ProElementDecimalsSet()` assigns the number of decimals to be used for the double value of an element in the feature. The double value is used in the feature dimension related to this element.

Note

Use the function `ProElementDecimalsSet()` before using the function `ProElementDoubleSet()` to ensure that the double value is assigned with the correct number of decimal places.

The following functions show how options are constructed and set for `ProElement*Get()` functions.

The function `ProElementwstroptsAlloc()` allocates the options used to retrieve wide string values.

The function `ProElementwstroptsExpressionsSet()` sets the options to retrieve values as expressions or relations, if they exist, instead of a string representations of the actual value. This function is applicable to nominal double and integer value elements only.

The function `ProElementwstroptsSignoptionsSet()` sets the options to retrieve values with special sign formatting (+/-), etc. This function is applicable to nominal double and integer value elements only.

The function `ProElementwstroptsFree()` frees the options used to retrieve string values.

The function `ProElementWstringGet()` returns a string value representation for the element. The function allows, optionally, a `ProElementWstrOpts()` structure that dictates the format of the output. Use the function `ProWstringFree()` to free this string.

The function `ProElementWstringSet()` sets the string value for the element.

The function `ProElementStringGet()` returns an ASCII string value representation for the element. The inputs for this function are the element and options for how the string should be obtained. The output is the ASCII string value. Free this string using `ProStringFree()`.

The function `ProElementStringSet()` sets the ASCII string value for the element.

The function `ProElementSpecialvalueGet()` returns the pointer representation for the element and the function `ProElementSpecialvalueSet()` sets the pointer representation for the element.

The function `ProElementBooleanGet()` returns the boolean representation for the element and the function `ProElementBooleanSet()` sets the boolean value for the element.

The function `ProElementTransformGet()` returns the transform representation for the element and the function `ProElementTransformSet()` sets the transform value for the element.

Feature Element Diagnostics

Functions Introduced:

- **ProElementDiagnosticsCollect()**
- **ProElemdiagnosticProarrayFree()**
- **ProElemdiagnosticSeverityGet()**
- **ProElemdiagnosticMessageGet()**
- **ProElemdiagnosticFree()**
- **ProReferenceDiagnosticsCollect()**

The function `ProElementDiagnosticsCollect()` obtains a `ProArray` of diagnostics for the element. These diagnostics include warnings and errors regarding the value of the element within the context of the feature and remainder of the element tree. Use the function `ProElemdiagnosticProarrayFree()` to free the `ProArray` of diagnostic items.

The function `ProElemdiagnosticSeverityGet()` returns the severity of the diagnostic item for the element.

The function `ProElemdiagnosticMessageGet()` obtains the message describing the diagnostic item for the element. This message is in the user's localized language.

The function `ProElemdiagnosticFree()` frees the diagnostic item for the element.

The function `ProReferenceDiagnosticsCollect()` obtains a `ProArray` of diagnostics for the reference element. These diagnostics include warnings and errors regarding the state of the reference element within the context of the feature.

Calling ProFeatureCreate()

Function introduced:

- **ProFeatureCreate()**

The syntax of `ProFeatureCreate()` is as follows:

```
ProError ProFeatureCreate (
  ProSelection             model,       /* (In)  The part on which the
                                                  feature is being
                                                  created. If the feature
                                                  is created in an
                                                  assembly, you must
                                                  specify the component
                                                  path. */
  ProElement               elemtree,    /* (In)  The element tree. */
  ProFeatureCreateOptions  options[],   /* (In)  An array of user
                                                  options. */
  int                      num_opts,    /* (In)  The number of options
                                                  in the options array. */
  ProFeature              *p_feature,   /* (Out) The feature handle. */
  ProErrorlist            *p_errors     /* (Out) The list of errors. */
)
```

The first input argument to `ProFeatureCreate()` identifies the solid that is to contain the new feature. This is expressed in the form of a `ProSelection` object.

> **Note**
>
> The `ProSelection` object input to `ProFeatureCreate()`, and all the `ProSelection` objects assigned to elements in the feature element tree, must all refer to the same root assembly.

The second input argument is the `ProElement` object that forms the root of the feature element tree.

The next two inputs are an array of creation options and the size of the array. The creation options specify what `ProFeatureCreate()` should do if the element tree is incomplete, or if the geometry cannot be constructed. Each option is one of the following values of the enumerated type `ProFeatureCreateOptions`:

- `PRO_FEAT_CR_NO_OPTS`—No options were chosen.
- `PRO_FEAT_CR_DEFINE_MISS_ELEMS`—Prompt the user to complete the feature if any elements are missing.
- `PRO_FEAT_CR_INCOMPLETE_FEAT`—Create the feature, even if some

elements are missing. The feature will appear in the Creo Parametric feature list and model tree, but will not be used in regeneration.

- PRO_FEAT_CR_FIX_MODEL_ON_FAIL—If the feature geometry cannot be constructed, prompt the user to fix the problem.

If no options are needed, you can set the array to NULL, and the size to zero. If you do not specify any options, ProFeatureCreate() fails and reports errors if the element tree is incomplete, or if the geometry cannot be constructed.

To check whether a feature is incomplete, use the function ProFeatureIsIncomplete().

The next argument is an output that provides a ProFeature object that identifies the newly created feature.

The final argument is an output that reports errors found in the feature element tree. This is designed as an aid to application developers because it is reporting errors that occur only as a result of incorrect application code; it is not designed as a way of reporting errors to the Creo Parametric user. The errors are written to a structure called ProErrorlist whose declaration, in ProItemerr.h, is as follows:

```
typedef struct
{
   ProItemerror         *error_list;
   int                  error_number;
} ProErrorlist;
  typedef struct
  {
    int                  err_item_id;
    ProErritemType       err_item_type;
    ProError             error;
  } ProItemerror;
    typedef enum ProErritemTypes
    {
      PRO_ERRITEM_NONE = -1,
      PRO_ERRITEM_FEATELEM = 1
    } ProErritemType;
```

The field error_list is an array of all the errors in the feature element tree found by ProFeatureCreate(). Each error has a value expressed in terms of the standard Creo Parametric TOOLKIT error type ProError, and can refer to an element of a specified identifier, or be a more general error.

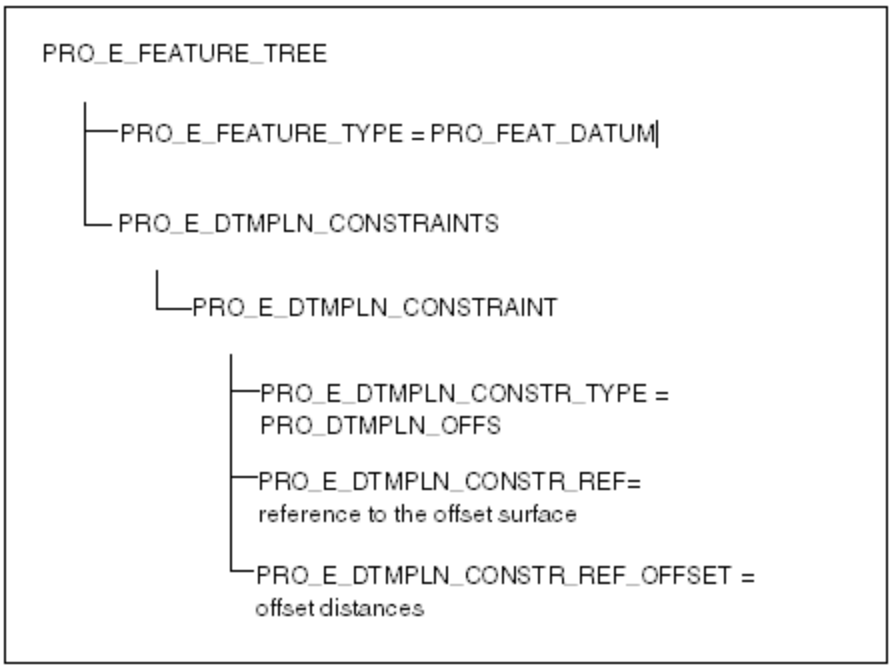
Note

- There are many useful utilities located in the Creo Parametric TOOLKIT sample code under the Creo Parametric TOOLKIT loadpoint. Utilities such as `ProUtilElementtreePrint()` are particularly useful when building and debugging element trees.
- From Pro/ENGINEER Wildfire 5.0 onward, the function `ProFeatureCreate()` has been deprecated. Instead, use the function `ProFeatureWithoptionsCreate()` with its input argument *flags* set to `PRO_REGEN_NO_FLAGS` for the equivalent behavior. `ProFeatureWithoptionsCreate()` has been described in detail in the Manipulating Features based on Regeneration Flags on page 141 section in the Core: Features on page 131 chapter.

Example of Complete Feature Creation

This section illustrates all the techniques explained so far in this chapter by showing the code required to create a datum plane in a part or an assembly. The datum plane created here is offset from a plane surface, and therefore has the following element tree:

Example Element Tree: Offset Datum Plane



(The full element tree for datum plane features is described in the chapter Element Trees: Datum Features on page 804.)

In the code examples that follow, no checks of function return statuses are shown, for clarity. No variable declarations are shown, but the style of the code samples should make these self-explanatory.

The example assumes that the datum plane is being created by a utility function that has two inputs describing the offset surface as a `ProSelection` relative to the current part or assembly, and the offset distance, as follows:

```
int ProDatumPlaneCreate (ProSelection offset_surface,
                         double       offset_dist);
```

The first step is to create the element that forms the root of the element tree. This element has the element ID `PRO_E_FEATURE_TREE` but has no value, so it can be created simply by a call to `ProElementAlloc()`:

```
/*--------------------------------------------------------------------*\
    Create the root of the element tree.
\*--------------------------------------------------------------------*/
    ProElementAlloc (PRO_E_FEATURE_TREE, &elem_tree);
```

The first element inside the root is, as for all features, the feature type. Its ID is `PRO_E_FEATURE_TYPE`, and it has the single value `PRO_FEAT_DATUM`. To set the value, you must first create a `ProValue` object of type integer.

```
/*--------------------------------------------------------------------*\
    Allocate the feature type element.
\*--------------------------------------------------------------------*/
    ProElementAlloc (PRO_E_FEATURE_TYPE, &elem_ftype);
/*--------------------------------------------------------------------*\
    Set the value of the feature type element.
\*--------------------------------------------------------------------*/
    value_data.type = PRO_VALUE_TYPE_INT;
    value_data.v.i = PRO_FEAT_DATUM;
    ProValueAlloc (&value);
    ProValueDataSet (value, &value_data);
    ProElementValueSet (elem_ftype, value);
/*--------------------------------------------------------------------*\
    Add the feature type element as a child of the tree root.
\*--------------------------------------------------------------------*/
    ProElemtreeElementAdd (elem_tree, NULL, elem_ftype);
```

The next element is simply the holder for the datum plane constraints, and this in turn contains a single constraint element (to be used for the offset constraint).

```
/*--------------------------------------------------------------------*\
    Add a PRO_E_DTMPLN_CONSTRAINTS element to the root of the
    tree.
\*--------------------------------------------------------------------*/
    ProElementAlloc (PRO_E_DTMPLN_CONSTRAINTS, &elem_consts);
    ProElemtreeElementAdd (elem_tree, NULL, elem_consts);
/*--------------------------------------------------------------------*\
    Add a PRO_E_DTMPLN_CONSTRAINT element to the constraints
    element.
```

```
\*-------------------------------------------------------------*/
    ProElementAlloc (PRO_E_DTMPLN_CONSTRAINT, &elem_offset);
    ProElemtreeElementAdd (elem_consts, NULL, elem_offset);
```

Inside the single constraint element, add an element of type PRO_E_DTMPLN_CONSTR_TYPE that specifies the constraint type to be PRO_DTMPLN_OFFS.

```
/*-------------------------------------------------------------*\
    Allocate the constraint type element.
\*-------------------------------------------------------------*/
    ProElementAlloc (PRO_E_DTMPLN_CONSTR_TYPE,
                     &elem_const_type);
/*-------------------------------------------------------------*\
    Set its value to be PRO_DTMPLN_OFFS.
\*-------------------------------------------------------------*/
    value_data.type = PRO_VALUE_TYPE_INT;
    value_data.v.i = PRO_DTMPLN_OFFS;
    ProValueAlloc (&value);
    ProValueDataSet (value, &value_data);
    ProElementValueSet (elem_const_type, value);
/*-------------------------------------------------------------*\
    Add it as a member of the constraint element.
\*-------------------------------------------------------------*/
    ProElemtreeElementAdd (elem_offset, NULL, elem_const_type);
```

Also in the constraint element, you need an element to identify the reference plane surface, PRO_E_DTMPLN_CONSTR_REF, with value type ProSelection.

```
/*-------------------------------------------------------------*\
    Allocate the offset reference element.
\*-------------------------------------------------------------*/
    ProElementAlloc (PRO_E_DTMPLN_CONSTR_REF, &elem_offset_ref);
/*-------------------------------------------------------------*\
    Set its value to be the ProSelection for the offset
    reference surface.
\*-------------------------------------------------------------*/
    value_data.type = PRO_VALUE_TYPE_SELECTION;
    value_data.v.r = offset_surface;
    ProValueAlloc (&value);
    ProValueDataSet (value, &value_data);
    ProElementValueSet (elem_offset_ref, value);
/*-------------------------------------------------------------*\
    Add it to the constraint element.
\*-------------------------------------------------------------*/
    ProElemtreeElementAdd (elem_offset, NULL, elem_offset_ref);
```

Finally, you need an element of type PRO_E_DTMPLN_CONSTR_REF_OFFSET to contain the double value of the offset distance.

```
/*-------------------------------------------------------------*\
    Allocate the offset distance element.
\*-------------------------------------------------------------*/
    ProElementAlloc (PRO_E_DTMPLN_CONSTR_REF_OFFSET,
                     &elem_offset_dist);
/*-------------------------------------------------------------*\
```

```
    Set its value to be the offset distance.
\*--------------------------------------------------------------------*/
    value_data.type = PRO_VALUE_TYPE_DOUBLE;
    value_data.v.d = offset_dist;
    ProValueAlloc (&value);
    ProValueDataSet (value, &value_data);
    ProElementValueSet (elem_offset_dist, value);
/*--------------------------------------------------------------------*\
    Add it to the constraint element.
\*--------------------------------------------------------------------*/
    ProElemtreeElementAdd (elem_offset, NULL, elem_offset_dist);
```

The element tree is complete.

The next step is to set up a `ProSelection` object that refers to the solid in which you will create the datum plane.

You have the information about the context, in the form of the `ProSelection` object, for the offset surface that was an input to the function you are writing. The component path you need is the same one used to specify that surface. The solid to contain the new feature is the one that owns the offset surface. Therefore, you can build the `ProSelection` object for it as follows:

```
/*--------------------------------------------------------------------*\
    Get the component path for the offset surface.
\*--------------------------------------------------------------------*/
    ProSelectionAsmcomppathGet (offset_surface, &comppath);
/*--------------------------------------------------------------------*\
    Get the model item for the offset surface.
\*--------------------------------------------------------------------*/
    ProSelectionModelitemGet (offset_surface, &surf_modelitem);
/*--------------------------------------------------------------------*\
    Make a ProModelitem that refers to the owner of the offset
    surface.
\*--------------------------------------------------------------------*/
    ProMdlToModelitem (surf_modelitem.owner, &model_modelitem);
/*--------------------------------------------------------------------*\
    Make a ProSelection for the solid that will contain the
    new feature.
\*--------------------------------------------------------------------*/
    ProSelectionAlloc (&comppath, &model_modelitem, &featsel);
```

If the offset surface belongs to a part in a current assembly, and your function is required to add the datum plane not to the part but to the assembly, the code above would be replaced by this:

```
/*--------------------------------------------------------------------*\
    Get the component path for the offset surface.
\*--------------------------------------------------------------------*/
    ProSelectionAsmcomppathGet (offset_surface, &comppath);
/*--------------------------------------------------------------------*\
    Make a ProModelitem that refers to the root of the assembly.
\*--------------------------------------------------------------------*/
    ProMdlToModelitem (comppath.owner, &model_modelitem);
```

```
/*--------------------------------------------------------------------*\
    Make a ProSelection for the root of the assembly.
\*--------------------------------------------------------------------*/
    ProSelectionAlloc (NULL, &model_modelitem, &featsel);
```

Finally, call `ProFeatureCreate()`.

```
/*--------------------------------------------------------------------*\
    Create the datum plane feature.
\*--------------------------------------------------------------------*/
    ProFeatureCreate (featsel, elem_tree, NULL, 0, &feature,
                      &errors);
```

Feature Inquiry

Functions introduced:

- **ProFeatureElemtreeCreate()**
- **ProFeatureElemtreeExtract()**
- **ProFeatureElemtreeFree()**
- **ProFeatureElemValueGet()**
- **ProFeatureElemValuesGet()**
- **ProFeatureElemDimensionIdGet()**
- **ProFeatureElemIsVisible()**
- **ProFeatureIsIncomplete()**
- **ProFeatureElemIsIncomplete()**

This section describes how to extract the element tree from a feature and analyze it. To find out how to inquire about the feature as a whole and its role in the owning solid, see the section Feature Inquiry on page 785.in the Core: Features on page 131.

The function `ProFeatureElemtreeCreate()` creates a copy of the feature element tree that describes the contents of a specified feature in the Creo Parametric database. It is applicable only to those feature types that can be created using `ProFeatureCreate()` (as described in Overview of Feature Creation on page 765). The tree can then be analyzed using the read-access functions, such as `ProElement*Get()`,`ProElement*Visit()`, and `ProElementArrayCount()` described in the sections Feature Elements on page 774 and Feature Element Paths on page 772.

Note

The function `ProFeatureElemtreeCreate()` has been deprecated as it does not provide options to resolve the paths of external references of the feature in case of assemblies. Use `ProFeatureElemtreeExtract()` instead.

The function `ProFeatureElemtreeExtract()` creates a copy of the feature element tree of a specified feature. It also provides options to resolve the paths of external references of the feature in case of assemblies.

Use the function `ProFeatureElemtreeFree()` to free a copy of the feature element tree extracted using `ProFeatureElemtreeCreate()` `ProFeatureElemtreeExtract()`. The function `ProElementFree()` does not free all of the feature-specific runtime data associated with the element tree, and thus can result in a memory leak for certain features.

Instead of copying the entire element tree to analyze it, you can extract information about particular elements directly from the feature. The remaining functions in this section serve that purpose.

The function `ProFeatureElemValueGet()` provides the value of a single-valued element specified by the `ProFeature` object and a `ProElempath`. The function `ProFeatureElemValuesGet()` provides the values of a multivalued element in a feature.

The function `ProFeatureElemDimensionIdGet()` gives you the integer identifier of the dimension in the Creo Parametric database used to define the value of the specified single-valued element.

The function `ProFeatureElemIsVisible()` distinguishes elements added to the tree by Creo Parametric for internal reasons only, and are neither defined as needed for creation of that type of feature, nor otherwise documented.

The function `ProFeatureIsIncomplete()` identifies features in the Creo Parametric database whose element trees are still incomplete. Such a feature can arise by using the option `PRO_FEAT_CR_INCOMPLETE_FEAT` when calling `ProFeatureCreate()`, and does not give rise to geometry until completed. If a feature is incomplete, you can find out which element in its tree is at fault using the function `ProFeatureElemIsIncomplete()`. Its input is a `ProFeature` and a `ProElempath`.

Feature Redefine

Function introduced:

- **ProFeatureRedefine()**

The function `ProFeatureRedefine()` enables you to redefine a feature. It is applicable only to those feature types that can be created using `ProFeatureCreate()`.

The function `ProFeatureRedefine()` has arguments for the create options and for the resulting element errors, like those for `ProFeatureCreate()`

To Redefine a Feature

1. Call `ProFeatureElemtreeExtract()` to get a copy of the element tree.
2. Analyze and modify the tree using functions `ProFeatureElem*()`, `ProElement*()`, and `ProElempath*()`
3. Call `ProFeatureRedefine()` to replace the old element tree with the new one.

XML Representation of Feature Element Trees

Creo Parametric TOOLKIT offers the capability of representing most feature element trees in the XML format. The XML representation of element trees simplifies the procedure of using Creo Parametric TOOLKIT access to Creo Parametric features. The Creo Parametric TOOLKIT API's that access the XML for element trees are capable of converting the XML content to and from an element tree structure (`ProElement`.)

In Pro/ENGINEER Wildfire 2.0, PTC recommends the use of XML representation for the core Pro/ENGINEER features and for the new sheetmetal wall features. Manufacturing features support is not available in this release.

Introduction to Feature Element Trees in XML

In Creo Parametric TOOLKIT, the access to Creo Parametric features is available using a generic data-structure called element tree. You can use the element tree APIs to query, modify, and redefine the Creo Parametric features. To simplify the access to Creo Parametric feature using Creo Parametric TOOLKIT, the concept of XML representation of the element trees has been introduced. The XML representation shows the details of the element tree in text format. The XML files are easy to edit and you can recycle them back to create or to redefine features. All the Creo Parametric TOOLKIT supported features have support for their respective XML representations.

The function `ProElemtreeWrite()` writes the XML version of the element tree in a text file and the function `ProElemtreeFromXMLCreate()` is used to convert the XML version from a text file to the Creo Parametric TOOLKIT native element trees.

Functions Introduced:

- **ProElemtreeWrite()**
- **ProElemtreeFromXMLCreate()**
- **ProXmlerrorlistProarrayFree()**

The function `ProElemtreeWrite()` writes the feature element tree in XML format to the specified file. This function uses the Creo Parametric TOOLKIT native element tree as an input. The type of output required should be specified. Currently only XML format for the output is supported. The only output argument is the name of the output file included with the location and the extension.

The function `ProElemtreeFromXMLCreate()` reads in the XML file and converts into a corresponding Creo Parametric TOOLKIT native element tree. This function takes the name with location of the output XML file as an input and returns the generated element tree as the output.

The created element tree can be used as an input to `ProFeatureCreate` or `ProFeatureRedefine`.

In case of error in execution of this function, an error objects is populated and returned. This error object is a ProArray of errors. It contains the information like type of the error, path of the XML file, line number, column number, and the error message. The memory for the error object is allocated by the function is required to be freed in the Creo Parametric TOOLKIT application using `ProXmlerrorlistProarrayFree()`. The function has three levels of error on the basis of the severity warning, general error and fatal error. Upon encountering warning and general errors, the function continues parsing the input XML till the end and collects all the errors in the error object. For a fatal error, the function stops the parsing and returns immediately.

Validation Using XML Schema

XML schemas are used for validation of element tree which is used for feature creation. Element tree in the form of XML representations are validated and passed to `ProFeatureCreate()` and `ProFeatureRedefine()`.

To use the Schema validation, the following config option must be set

The XML Schema defines the structure and data type for XML. The element name is compared with the element specified in schema for validation. In the case of compound elements, validation is done for name, order and number of the sub-

elements. The Schema validates the accuracy of data to be passed as it supports the data type of the element. The Schema also checks whether the element is optional or compulsory.

The location of the Schema is `<install_dir>/proe/xmlelem/schema/`

`ProTKFeature.xsd` contains the representation of different feature element.

`ProTKObjects.xsd` contains the representation of basic and Creo Parametric TOOLKIT objects.

The schema validation of element tree takes place if the config option `enable_protk_xml_schema` is set to "`yes`" before passing XML file to `ProElemtreeFromXMLCreate()`.

If the validation fails, the corresponding error is populated into the error object which can be printed out and can be used to correct the input XML file. The error object contains the line number, column number and the error message as debug information.

The following is an example of Schema for Solidify Feature:

```
<xsd:sequence>
      <xsd:element name="PRO_E_FEATURE_TYPE"
type="StringData"
      minOccurs="1" maxOccurs="1"/>
      <xsd:element name="PRO_E_FEATURE_FORM" type="StringData"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="PRO_E_PATCH_QUILT" type="Selection"
        minOccurs="1" maxOccurs="1"/>
      <xsd:element name="PRO_E_PATCH_TYPE" type="IntegerData"
        minOccurs="0" maxOccurs="1"/>
      <xsd:element name="PRO_E_PATCH_MATERIAL_SIDE" type="IntegerData"
        minOccurs="0" maxOccurs="1"/>
      <xsd:element name="PRO_E_STD_FEATURE_NAME" type="StringData"
        minOccurs="0" maxOccurs="1"/>
     </xsd:sequence>
    <!-- End Solidify -->
```

Following code demonstrate how to use the error object to print the information

```
status = ProElemtreeFromXMLCreate ( input_file, &elemtree, &errors );
if ( ( status == PRO_TK_GENERAL_ERROR ) || ( status ==   PRO_TK_BAD_CONTEXT ))
{
  printf ("ProElemtreeFromXMLCreate FAILURE ::  %d \n", status );
  status = ProArraySizeGet ( errors, &size_errors );
  printf("Size of the Error Object is %d \n", size_errors);
  printf("Content of the Error Object are \n");
  for ( error_count = 0; error_count < size_errors; error_count++ )
    {
      printf("Error No  = %d\n", error_count);
      printf("\t+ Error Type  = %d\n", errors[error_count].error_type   );
      if ( status == PRO_TK_GENERAL_ERROR )
        {
          printf("\t+ Error Path  = %s\n",
```

```
        ProWstringToString ( temp_string, errors[error_count].error_source ));
        printf("\t+ Line  No  = %d\n", errors[error_count].line_number);
        printf("\t+ Column  No  = %d\n", errors[error_count].column_number);
         }
       else if ( status == PRO_TK_BAD_CONTEXT )
         {
        printf("\t+ Element Sr No  = %d\n",
errors[error_count].line_number);
        printf("\t+ Element ID     = %d\n",
errors[error_count].column_number);
         }
   ProWstringLengthGet ( errors[error_count].message,
&string_length );
   temp_message_string = (char *) calloc ( 1, (string_length+1) * sizeof (char) );
   printf("\t+ Error Message  = %s\n",
     ProWstringToString ( temp_message_string,
errors[error_count].message ));
    free ( temp_message_string );
    }
}
```

The list of features supported for XML schema validation are as follows:

Feature Type	Feature Name
PRO_FEAT_DATUM_AXIS	Datum Axis Feature
PRO_FEAT_DATUM	Datum Plane Feature
PRO_FEAT_CSYS	Datum Coordinate System Feature
PRO_FEAT_CURVE	Datum Curve Feature
PRO_FEAT_DATUM_POINT	Datum Point Feature
PRO_FEAT_CHAMFER	Chamfer Feature
PRO_FEAT_ROUND	Round Feature
PRO_FEAT_DRAFT	Draft Feature
PRO_FEAT_HOLE	Hole Feature
PRO_FEAT_WALL	Sheetmetal Flat or Flange Wall Feature
PRO_FEAT_COMPONENT	Assembly Component Feature
PRO_FEAT_DATUM_QUILT	Surface Merge Feature
PRO_FEAT_PATCH	Solidify Feature of the type Patch
PRO_FEAT_GEN_MERGE	General Merge Feature
PRO_FEAT_SRF_MDL	Mirror or Move Feature
PRO_FEAT_PROTRUSION	Solidify Feature of the type Protrusion
PRO_FEAT_SHELL	Shell Feature
PRO_FEAT_FIRST_FEAT	First Feature
PRO_FEAT_CUT	Solidify Feature of the type Cut
PRO_FEAT_SLOT	Slot Feature
PRO_FEAT_GEOM_COPY	Copy Geometry Feature
PRO_FEAT_RIB	Rib Feature
PRO_FEAT_REPLACE_SURF	Replace Surface Feature

XML Representations for Common Elements

The following section gives details about the XML representation of the common elements. The elements can represent a primitive type e.g. an integer or some complex Creo Parametric TOOLKIT object e.g. `ProReference` or `ProCollection`. The purpose of this representation is to enable externalize these objects in a model specific text format. The examples of the XML representations are also provided for easier understanding.

Single Valued Element

A single valued element contains an element of type Integer, Double, String, or Boolean. Its value will be shown as the XML tags. For example:

```
<PRO_E_REV_ANGLE_FROM_VAL type="double">0.00</PRO_E_REV_ANGLE_FROM_VAL>
```

Empty or Optional Element

An empty element in XML does not have any data value, but just the opening and closing XML tags. Empty elements are ignored by `ProElemtreeFromXMLCreate()`. Unusedoptional elements in the feature's element tree appear as empty in the XML output from `ProElemtreeWrite()`.

```
<PRO_E_REV_ANGLE_FROM_REF type="selection" />
```

OR

```
<PRO_E_REV_ANGLE_FROM_REF type="selection"></PRO_E_REV_ANGLE_FROM_REF type>
```

XML Representation for ProSelection or ProReference

This represents the contents of the `ProSelection` or `ProReference` object. If the reference is currently not active or fully available, the reference is included. Consult the function `ProReferenceStatusGet()` for more information.The reference XML includes details like UV parameters and 3D point for specific elements that require them. The assembly component path will be available for references that include the assembly context.

```
<PRO_E_DPOINT_PLA_CONSTR_REF type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_STATUS type= “int”>PRO_REF_NOT_FOUND</PRO_XML_REFERENCE_STATUS>
<PRO_XML_REFERENCE_OWNER type="owner">COMP2.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">46</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_EDGE</PRO_XML_REFERENCE_TYPE>
<PRO_XML_ASMCOMP_PATH comppath="compound">
<PRO_XML_ASMCOMP_PATH_OWNER type="model">ASM_PNT.asm</PRO_XML_ASMCOMP_PATH_OWNER>
<PRO_XML_ASMCOMP_PATH_ARRAY type="array">
<PRO_XML_ASMCOMP_PATH_ITEM index="1">40</PRO_XML_ASMCOMP_PATH_ITEM>
     </PRO_XML_ASMCOMP_PATH_ARRAY>
     </PRO_XML_ASMCOMP_PATH>
<PRO_XML_UV_PARAM uv_parameter="array">
<PRO_XML_DOUBLE_VALUE type="u">0.000000</PRO_XML_DOUBLE_VALUE>
<PRO_XML_DOUBLE_VALUE type="v">0.000000</PRO_XML_DOUBLE_VALUE>
```

```
</PRO_XML_UV_PARAM>
</PRO_XML_REFERENCE>
</PRO_E_DPOINT_PLA_CONSTR_REF>
```

XML Representation for ProCollection

This represents the contents of the `ProCollection` object. Collection type, collection instructions, instruction types, references, reference types are some of the contents. These are classified into surface and curve collections. Surface collection represents a set of surfaces in the model governed by rules like seed and boundary. Curve collection represents a set of edges or curves in the model governed by rules like one-by-one, surface-loop etc. The instruction type for the particular collection identifies the rule.

Curve Collection

```
<PRO_E_STD_CURVE_COLLECTION_APPL type="collection">
  <PRO_XML_COLLECTION type="curve">
    <PRO_XML_COLLECTION_INSTRUCTIONS type="array">
   <PRO_XML_COLLECTION_INSTRUCTION type="compound">
   <PRO_XML_COLLECTION_INSTRUCTION_TYPE type="int">105</PRO_XML_COLLECTION_
INSTRUCTION_TYPE>
    <PRO_XML_CRVCOLL_REFS type="array">
    <PRO_XML_CRVCOLL_REF type="selection">
    <PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">CRV_COLLECTION_PART.prt
               </PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">703</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_QUILT</PRO_XML_REFERENCE_TYPE>
                   </PRO_XML_REFERENCE>
                 </PRO_XML_CRVCOLL_REF>
                 <PRO_XML_CRVCOLL_REF type="selection">
                     <PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">CRV_COLLECTION_PART.prt
               </PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">679</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_EDGE</PRO_XML_REFERENCE_TYPE>
                    </PRO_XML_REFERENCE>
                  </PRO_XML_CRVCOLL_REF>
            </PRO_XML_CRVCOLL_REFS>
      </PRO_XML_COLLECTION_INSTRUCTION>
   </PRO_XML_COLLECTION_INSTRUCTIONS>
 </PRO_XML_COLLECTION>
</PRO_E_STD_CURVE_COLLECTION_APPL>
```

Surface Collection

```
<PRO_E_STD_SURF_COLLECTION_APPL type="collection">
  <PRO_XML_COLLECTION type="surface">
   <PRO_XML_COLLECTION_INSTRUCTIONS type="array">
```

```
<PRO_XML_COLLECTION_INSTRUCTION type="compound">
   <PRO_XML_COLLECTION_INSTRUCTION_TYPE type="int">2
    </PRO_XML_COLLECTION_INSTRUCTION_TYPE>
   <PRO_XML_SRFCOLL_INCLUDE type="boolean">1</PRO_XML_SRFCOLL_INCLUDE>
   <PRO_XML_SRFCOLL_REFS type="array">
   <PRO_XML_SRFCOLL_REF type="compound">
   <PRO_XML_SRFCOLL_REFITEM_TYPE type="int">3</PRO_XML_SRFCOLL_REFITEM_TYPE>
   <PRO_XML_SRFCOLL_REFITEM type="selection">
   <PRO_XML_REFERENCE type="reference">
   <PRO_XML_REFERENCE_OWNER type="owner">
    SRF_COLLECTION_PART.prt</PRO_XML_REFERENCE_OWNER>
   <PRO_XML_REFERENCE_ID type="id">17</PRO_XML_REFERENCE_ID>
   <PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
  </PRO_XML_REFERENCE>
 </PRO_XML_SRFCOLL_REFITEM>
</PRO_XML_SRFCOLL_REF>
<PRO_XML_SRFCOLL_REF type="compound">
   <PRO_XML_SRFCOLL_REFITEM_TYPE type="int">4</PRO_XML_SRFCOLL_REFITEM_TYPE>
    <PRO_XML_SRFCOLL_REFITEM type="selection">
    <PRO_XML_REFERENCE type="reference">
    <PRO_XML_REFERENCE_OWNER type="owner">
    SRF_COLLECTION_PART.prt</PRO_XML_REFERENCE_OWNER>
    <PRO_XML_REFERENCE_ID type="id">26</PRO_XML_REFERENCE_ID>
    <PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE
 </PRO_XML_REFERENCE>
 </PRO_XML_SRFCOLL_REFITEM>
 </PRO_XML_SRFCOLL_REF>
 </PRO_XML_SRFCOLL_REFS>
 </PRO_XML_COLLECTION_INSTRUCTION>
 </PRO_XML_COLLECTION_INSTRUCTIONS>
 </PRO_XML_COLLECTION>
 </PRO_E_STD_SURF_COLLECTION_APPL>
```

Pointer Element

Most pointer elements are written with their value appearing as "**". Elements with this value type are not supported by `ProElemtreeFromXMLCreate()` (this element is not added to the created element tree.) The most common pointer element of this type is the Creo Parametric sketch element `PRO_E_SKETCHER`.

```
<PRO_E_SKETCHER type="pointer">**</PRO_E_SKETCHER>
```

Compound Element

A compound element represents a collection of different types of sub-elements. Following is an example in XML format.

```
<PRO_E_REV_ANGLE type="compound">
     <PRO_E_REV_ANGLE_FROM type="compound">
        <PRO_E_REV_ANGLE_FROM_TYPE type="int">262144</PRO_E_REV_ANGLE_FROM_TYPE>
        <PRO_E_REV_ANGLE_FROM_VAL type="double">0.00</PRO_E_REV_ANGLE_FROM_VAL>
        <PRO_E_REV_ANGLE_FROM_REF type="selection" />
```

```
        <PRO_E_REV_ANGLE_FROM_LIMIT type="int">0</PRO_E_REV_ANGLE_FROM_LIMIT>
        </PRO_E_REV_ANGLE_FROM>
        <PRO_E_REV_ANGLE_TO type="compound">
        <PRO_E_REV_ANGLE_TO_TYPE type="int">4194304</PRO_E_REV_ANGLE_TO_TYPE>
        <PRO_E_REV_ANGLE_TO_VAL type="double">120.00</PRO_E_REV_ANGLE_TO_VAL>
        <PRO_E_REV_ANGLE_TO_REF type="selection" />
        <PRO_E_REV_ANGLE_TO_LIMIT type="int">0</PRO_E_REV_ANGLE_TO_LIMIT>
   </PRO_E_REV_ANGLE_TO>
</PRO_E_REV_ANGLE>
```

Array Element

An array element represents a collection of elements of the same type. Following is an example of an array of PRO_E_RNDCH_RADIUS elements in XML format.

```
<PRO_E_RNDCH_RADII type="array">
     <PRO_E_RNDCH_RADIUS type="compound">
            <PRO_E_STD_POINT_COLLECTION_APPL type="selection">
                  <PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">K01_X_COLLECTION_PART.prt
            </PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">1268</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_CRV_END</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_STD_POINT_COLLECTION_APPL>
<PRO_E_RNDCH_LEG1 type="compound">
<PRO_E_RNDCH_LEG_TYPE type="int">1</PRO_E_RNDCH_LEG_TYPE>
<PRO_E_RNDCH_LEG_VALUE type="double">36.00</PRO_E_RNDCH_LEG_VALUE>
<PRO_E_RNDCH_REFERENCE_EDGE type="selection" />
<PRO_E_RNDCH_REFERENCE_POINT type="selection" />
</PRO_E_RNDCH_LEG1>
<PRO_E_RNDCH_LEG2 type="compound">
<PRO_E_RNDCH_LEG_TYPE type="int">0</PRO_E_RNDCH_LEG_TYPE>
<PRO_E_RNDCH_LEG_VALUE type="double">0.00</PRO_E_RNDCH_LEG_VALUE>
<PRO_E_RNDCH_REFERENCE_EDGE type="selection" />
<PRO_E_RNDCH_REFERENCE_POINT type="selection" />
</PRO_E_RNDCH_LEG2>
</PRO_E_RNDCH_RADIUS>
<PRO_E_RNDCH_RADIUS type="compound">
<PRO_E_STD_POINT_COLLECTION_APPL type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">K01_X_COLLECTION_PART.prt
           </PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">1268</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_CRV_START</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_STD_POINT_COLLECTION_APPL>
<PRO_E_RNDCH_LEG1 type="compound">
<PRO_E_RNDCH_LEG_TYPE type="int">1</PRO_E_RNDCH_LEG_TYPE>
<PRO_E_RNDCH_LEG_VALUE type="double">55.39</PRO_E_RNDCH_LEG_VALUE>
<PRO_E_RNDCH_REFERENCE_EDGE type="selection" />
```

```
<PRO_E_RNDCH_REFERENCE_POINT type="selection" />
</PRO_E_RNDCH_LEG1>
<PRO_E_RNDCH_LEG2 type="compound">
<PRO_E_RNDCH_LEG_TYPE type="int">0</PRO_E_RNDCH_LEG_TYPE>
<PRO_E_RNDCH_LEG_VALUE type="double">0.00</PRO_E_RNDCH_LEG_VALUE>
<PRO_E_RNDCH_REFERENCE_EDGE type="selection" />
<PRO_E_RNDCH_REFERENCE_POINT type="selection" />
</PRO_E_RNDCH_LEG2>
</PRO_E_RNDCH_RADIUS>
</PRO_E_RNDCH_RADII>
```

Multivalued Element

The multi-valued element represents an array of same types of elements. The difference between the array and multi-valued elements lies in the Creo Parametric TOOLKIT handling of these elements. In the XML format both the representations are similar.

```
<PRO_E_SHELL_SRF type="multivalue">
     <PRO_E_SHELL_SRF_MULTI type="selection">
            <PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL_REDEF.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">76</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_SHELL_SRF_MULTI>
<PRO_E_SHELL_SRF_MULTI type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL_REDEF.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">72</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_SHELL_SRF_MULTI>
<PRO_E_SHELL_SRF_MULTI type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL_REDEF.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">68</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_SHELL_SRF_MULTI>
</PRO_E_SHELL_SRF>
```

Tips for Recycling XML Output of Element Trees

The easiest way to create an XML file for any feature element tree is to call `ProFeatureElemtreeExtract()` followed by `ProElemtreeWrite()`. The pt_userguide application includes this capability. While running this application, select a feature. Click the right mouse button and choose **Export XML**. The output XML produced can be used and reused with modifications by passing it to `ProElemtreeFromXMLCreate()` to create an element tree. The

resulting element tree can be used for new feature creation or redefinition of an existing feature. The following tips would help easier reuse and understanding failures.

- Creo Parametric models can have only a single feature with a specific name - before using the XML for creating a new feature, modify the value of the element `PRO_E_STD_FEATURE_NAME` appropriately. Since, feature name is an optional element, the XML tag for this element can be deleted from the XML file to promote reuse.
- The function `ProElemtreeWrite()` writes all of the elements in the input Creo Parametric TOOLKIT native element tree. Optional elements with empty data within XML tags from the XML file can be removed before reusing the XML file. It is also permitted to keep the optional elements as they are in the XML, but these elements are ignored by `ProElemtreeFromXMLCreate()`.
- `ProElemtreeFromXMLCreate()` converts the XML file to an element tree without validating that the data is appropriate for that feature type (for example, ensuring that the correct type is selected for references.) The error is detected when the element tree is used by `ProFeatureCreate()` or `ProFeatureRedefine()`.
- `ProElemtreeFromXMLCreate()` supports some enumerated symbols or the actual integer values. Use the output XML file as a guide to see which enumerated type names are accepted.
- The tag names used in the XML representation are derived from the list of `ProElemId` from `ProElemId.h`.These names must be used exactly as found in the `ProElemId.h` file.
- XML files created by one version of Creo Parametric are not guaranteed to be usable by other Creo Parametric versions, because the structure of the element tree also change between versions.

A Sample Element Tree in XML for a Shell Feature

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML File for shell feature -->
<PRO_E_FEATURE_TREE AppName="Creo Parametric" AppVersion="2003440"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ProTKFeature.xsd" type="compound">
  <PRO_E_FEATURE_TYPE type="int">PRO_FEAT_SHELL</PRO_E_FEATURE_TYPE>
  <PRO_E_STD_FEATURE_NAME type="wstring">SHELL</PRO_E_STD_FEATURE_NAME>
  <PRO_E_SHELL_SRF type="multivalue">
       <PRO_E_SHELL_SRF_MULTI type="selection">
              <PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">76</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
```

```
</PRO_XML_REFERENCE>
</PRO_E_SHELL_SRF_MULTI>
</PRO_E_SHELL_SRF>
<PRO_E_SHELL_THICK type="double">4.02</PRO_E_SHELL_THICK>
<PRO_E_SHELL_FLIP type="int">1</PRO_E_SHELL_FLIP>
       <PRO_E_ST_SHELL_LOCL_LIST type="array">
              <PRO_E_ST_SHELL_LOCL_CMPD type="compound">
<PRO_E_ST_SHELL_SPEC_SRF type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">53</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_ST_SHELL_SPEC_SRF>
<PRO_E_ST_SHELL_SPEC_THCK type="double">10.00</PRO_E_ST_SHELL_SPEC_THCK>
</PRO_E_ST_SHELL_LOCL_CMPD>
<PRO_E_ST_SHELL_LOCL_CMPD type="compound">
<PRO_E_ST_SHELL_SPEC_SRF type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">66</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_ST_SHELL_SPEC_SRF>
<PRO_E_ST_SHELL_SPEC_THCK type="double">9.00</PRO_E_ST_SHELL_SPEC_THCK>
</PRO_E_ST_SHELL_LOCL_CMPD>
<PRO_E_ST_SHELL_LOCL_CMPD type="compound">
<PRO_E_ST_SHELL_SPEC_SRF type="selection">
<PRO_XML_REFERENCE type="reference">
<PRO_XML_REFERENCE_OWNER type="owner">X_SHELL.prt</PRO_XML_REFERENCE_OWNER>
<PRO_XML_REFERENCE_ID type="id">74</PRO_XML_REFERENCE_ID>
<PRO_XML_REFERENCE_TYPE type="protype">PRO_SURFACE</PRO_XML_REFERENCE_TYPE>
</PRO_XML_REFERENCE>
</PRO_E_ST_SHELL_SPEC_SRF>
<PRO_E_ST_SHELL_SPEC_THCK type="double">15.00</PRO_E_ST_SHELL_SPEC_THCK>
</PRO_E_ST_SHELL_LOCL_CMPD>
</PRO_E_ST_SHELL_LOCL_LIST>
</PRO_E_FEATURE_TREE>
```

The following example shows how to use `ProElemtreeWrite()` to export a feature element tree to XML.

```
#include <ProToolkit.h>
#include <ProMenuBar.h>
#include <ProMdl.h>
#include <ProSelection.h>
#include <ProFeature.h>
#include <ProWstring.h>
static
wchar_t MSGFIL[] = {'u','t','i','l','i','t','i','e','s','.','t','x','t','\0'};
/*====================================================================*\
FUNCTION: UserFeatXMLWrite
```

```
PURPOSE:  Write a feature's element tree to an XML file.
\*=====================================================================*/
int UserFeatXMLWrite (ProFeature* feat)
{
  ProElement elemtree;
  wchar_t wFilename [PRO_FILE_NAME_SIZE];
/*--------------------------------------------------------------------*\
Prompt for the filename
\*--------------------------------------------------------------------*/
  ProMessageDisplay ( MSGFIL, "USER Enter the XML file name:");
  if (ProMessageStringRead (PRO_FILE_NAME_SIZE, wFilename)
      != PRO_TK_NO_ERROR)
    return (0);
/*--------------------------------------------------------------------*\
Extract the element tree and convert it to XML
\*--------------------------------------------------------------------*/
  status = ProFeatureElemtreeExtract (feat,NULL,PRO_FEAT_EXTRACT_NO_OPTS,
                                      &elemtree);
  if ( status == PRO_TK_NO_ERROR )
  {
    status = ProElemtreeWrite (elemtree, PRO_ELEMTREE_XML, wFilename);
    ProElementFree (&elemtree);
  }
  return status;
}
```

33

Element Trees: References

This chapter describes functions that provide access to reference objects as an alternative for accessing the information as selections.

Overview of Reference Objects

Reference objects are an alternative method for representing a geometric reference in Creo Parametric. Geometric references are usually represented using the `ProSelection` structure. Since `ProSelection` is designed as the result of an interactively selected item, it lacks some capabilities to provide complete meaning as a geometric reference.

The opaque handle `ProReference` provides complete functionality for geometric referencing including functions to access multiple or missing references.

Reading References

Functions Introduced:

- **ProReferenceStatusGet()**
- **ProReferenceIsLocalcopy()**
- **ProReferenceTypeGet()**
- **ProReferenceIdGet()**
- **ProReferenceOwnerGet()**
- **ProReferenceOwnerMdlnameGet()**
- **ProReferenceOriginaltypeGet()**
- **ProReferenceOriginalidGet()**
- **ProReferenceOriginalownerGet()**
- **ProReferenceOriginalownerMdlnameGet()**
- **ProReferenceAsmcomppathGet()**
- **ProReferenceParamsGet()**
- **ProReferencePointGet()**
- **ProReferenceToSelection()**
- **ProSelectionToReference()**
- **ProReferencearrayToSelections()**
- **ProSelectionarrayFree()**
- **ProSelectionarrayToReferences()**
- **ProReferenceFree()**
- **ProReferencearrayFree()**

The function `ProReferenceStatusGet()` identifies the status of the reference. Typically references are of status `PRO_REF_ACTIVE`, indicating that the reference is available and its geometry is usable for construction of features. Other reference statuses include:

- `PRO_REF_MISSING` – The reference is to geometry that is inactive in the model. When reading the element tree of a feature that modifies the geometry it references (for example, a round of edge removes the edge) the references will have this status.
- `PRO_REF_NOT_FOUND` – Similar to `PRO_REF_MISSING`. The function indicates a reference that is critical to the feature that uses the function.
- `PRO_REF_FROZEN` – The reference is to geometry in a component frozen due to other missing references.
- `PRO_REF_FROZEN_PLACE`, `PRO_REF_SUPPRESSED`, `PRO_REF_EXCLUDED` – Not returned by `ProReferenceStatusGet()`.
- `PRO_REF_INVALID` – The reference is invalid in the context of the feature and the element in which it is used (for example, a non-linear edge used as direction reference for drafts or translations.)
- `PRO_REF_ALTERNATE` – The reference is using an alternate reference. The original reference information is available.

Note

Reference status is highly dependent on the model state at the time the reference was obtained. References obtained from a feature via `ProFeatureElemtreeExtract()` may become “Missing” or “Not found” due to changes in the geometry applied later in the feature list. To obtain the reference status for a given reference as it is seen by a feature, use `ProInsertModeActivate()` to revert the model to the state just after that feature is created.

- `PRO_REF_WARNING`—The referenced entity has a warning.

Note

References obtained from a feature that removes a geometric empty body can have a “Warning” or “Invalid” state. The statuses of the references are not stored in the models and are thrown at run time during feature creation, redefinition, or regeneration.

The function `ProReferenceIsLocalcopy()` identifies if the reference is a local copy of the external reference. If the reference is a local copy, the original reference information is available as well.

The function `ProReferenceTypeGet()` gets the type of handle that is referenced.

The function `ProReferenceIdGet()` gets the item identity of the reference handle.

The function `ProReferenceOwnerGet()` gets the `ProMdl` handle of the owner model for the reference. The output of these three functions provides access to the basic `ProModelitem` referenced by the `ProReference` handle. If the reference status is not `PRO_REF_ACTIVE`, some or all of this information may not be accessible (for example, the reference owner model is not accessible if the reference is to a geometry item in an unretrieved component). The function `ProReferenceOwnerMdlnameGet()` gets the reference owner name of the referenced geometry item, which may be available even if the model itself has not been retrieved.

The functions `ProReferenceOriginaltypeGet()`, `ProReferenceOriginalidGet()`, `ProReferenceOriginalownerGet()`, `ProReferenceOriginalownerMdlnameGet()` get the original properties of a geometric reference handle. These could be different from the actively used type if the reference has been backed up, copied locally, or replaced with an alternate.

The function `ProReferenceAsmcomppathGet()` gets the component path of a reference handle.

The function `ProReferenceParamsGet()` gets the u-v parameters of a reference handle.

The function `ProReferencePointGet()` gets the selected point of a reference handle.

The function `ProReferenceToSelection()` gets and allocates a `ProSelection` containing a representation for this reference. The output of this function is the resulting `ProSelection` handle. This selection is independently allocated and should be freed using `ProSelectionFree()`.

The function `ProSelectionToReference()` gets and allocates a `ProReference` containing a representation for this selection. The output of this function is the resulting `ProReference` handle. This reference is independently allocated and should be freed using `ProReferenceFree()`.

The function ProReferencearrayToSelections() converts a reference ProArray to a selection ProArray. The input arguments for this function are

- *references* – The ProArray of reference handles.
- *skip_unusable* – PRO_B_TRUE to skip the processing of missing references that cannot be valid selections. PRO_B_FALSE to process all references.

The output for this function is a selection that is a ProArray of selection handles. You can free this array using the function ProSelectionarrayFree().

The function ProSelectionarrayToReferences() converts a selection ProArray to a reference ProArray. ProArray of selection handles is given as the input and the ProArray of reference handles is the output. Free this array using ProReferencearrayFree().

The function ProReferenceFree() frees a reference handle.

The function ProReferencearrayFree() frees a reference ProArray. This function also free each ProReference handle using ProReferenceFree().

Modifying References

Functions Introduced:

- **ProReferenceAlloc()**
- **ProReferenceSet()**
- **ProReferenceParamsSet()**
- **ProReferencePointSet()**

The function ProReferenceAlloc() allocates a reference handle.

The function ProReferenceSet() sets the referenced model item and optionally component path for the reference handle.

The function ProReferenceParamsSet() sets the u-v parameters of a reference handle.

The function ProReferencePointSet() sets the selected point of a reference handle.

34

Element Trees: Datum Features

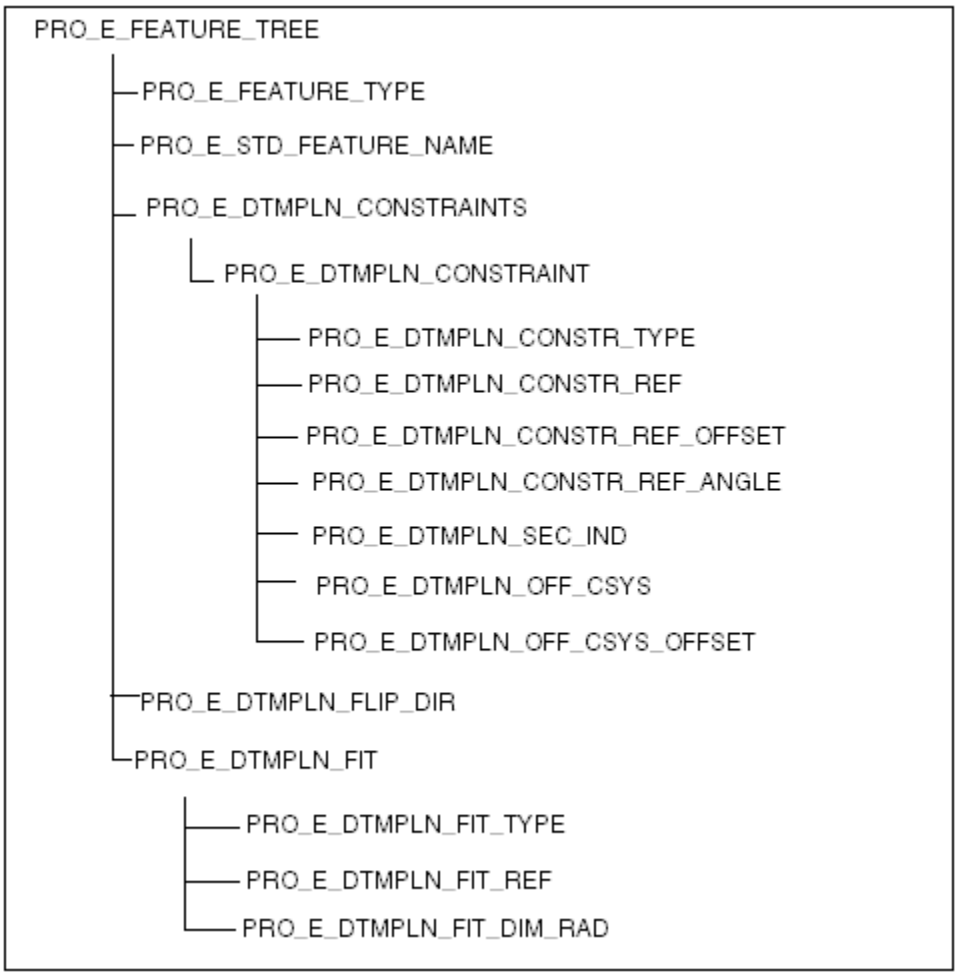
This chapter describes how to use the include files `ProDtmPln.h`, `ProDtmPnt.h`, `ProDtmAxis.h`, and `ProDtmCsys.h` to create datum features programmatically. The chapter on Element Trees: Principles of Feature Creation on page 764 provides necessary background for creating datum features; we recommend you read that material first.

Datum Plane Features

The element tree for a datum plane feature is documented in the header file `ProDtmPln.h`, and has a simple structure. Apart from the usual elements for the tree root and feature type, a datum plane contains the positioning constraints, an optional flip direction, and an optional fit type.

The constraints element `PRO_E_DTMPLN_CONSTRAINTS` is an array element that contains a `PRO_E_DTMPLN_CONSTRAINT` element for each constraint. Many elements forming the constraint element `PRO_E_DTMPLN_CONSTRAINT` are used only for certain constraint types, so any given datum plane may contain fewer elements than are shown in the tree. Similarly, all the elements forming the constraint element `PRO_E_DTMPLN_FIT` are not always essential.

The following figure shows the element tree for datum planes.



Many elements forming the constraint element `PRO_E_DTMPLN_CONSTRAINT` are used only for the following constraint types:

- PRO_E_DTMPLN_CONSTR_REF_OFFSET—Used if the constraint type is “offset.”
- PRO_E_DTMPLN_CONSTR_REF_ANGLE—Used if the constraint type is “angle.”
- PRO_E_DTMPLN_CONSTR_SEC_IND—Used if the constraint type is “section.”
- PRO_E_DTMPLN_OFF_CSYS—Used if the constraint type is “offset” and the reference is “Csys.”
- PRO_E_DTMPLN_OFF_CSYS_OFFSET—Used if the constraint type is “offset” and the reference is “Csys.”

Similarly, elements of the optional element PRO_E_DTMPLN_FIT are used for the following fit types:

- PRO_E_DTMPN_FIT_REF—Used if the fit type is not “default” or “fit.”
- PRO_E_DTMPLN_FIT_DIM_RAD—Used if the fit type is “fit radius.”

The following table describes the tree elements in detail:

Element ID	Element Name	Data Type	Valid Value
PRO_E_FEATURE_TYPE	Feature type	PRO_VALUE_TYPE_INT	PRO_FEAT_DATUM
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_DTMPLN_CONSTRAINTS	Constraints	Array	
PRO_E_DTMPLN_CONSTRAINT	Constraints	Compound	
PRO_E_DTMPLN_CONSTR_TYPE	Type	PRO_VALUE_TYPE_INT	See ProDtmplnConstrType
PRO_E_DTMPLN_CONSTR_REF	References	PRO_VALUE_TYPE_SELECTION	See Constraint Reference Types on page 807
PRO_E_DTMPLN_CONSTR_REF_OFFSET	Offset	PRO_VALUE_TYPE_DOUBLE	Any
PRO_E_DTMPLN_CONSTR_REF_ANGLE	Angle	PRO_VALUE_TYPE_DOUBLE	(-360.0, 360.0)
PRO_E_DTMPLN_SEC_IND	Section index	PRO_VALUE_TYPE_INT	[0, sec num - 1]
PRO_E_DTMPLN_OFF_CSYS	Offset coordinate system	PRO_VALUE_TYPE_INT	See ProDtmplnOffCsysAxis
PRO_E_DTMPLN_OFF_CSYS_OFFSET	Offset coordinate system value	PRO_VALUE_TYPE_DOUBLE	Any
PRO_E_DTMPLN_FLIP_DIR	Flip direction	PRO_VALUE_TYPE_INT	ProDtmplnFlipDir
PRO_E_DTMPLN_FIT	Fit	Compound	
PRO_E_DTMPLN_FIT_TYPE	Fit type	PRO_VALUE_TYPE_INT	ProDtmplnFitType

Element ID	Element Name	Data Type	Valid Value
PRO_E_DTMPLN_FIT_REF	Reference	PRO_VALUE_TYPE_SELECTION	See Fit Reference Types on page 808
PRO_E_DTMPLN_FIT_DTM_RAD	Datum radius	PRO_VALUE_TYPE_DOUBLE	>= 0.0

Constraint Reference Types

The following table does not describe the entire list of combinations of geometrical constraints that can be applied, or the rules for what geometry items they can refer to. These are partially documented in Note 1 of the elements table in ProDtmPln.h, which includes the following information:

Constraint Type	Valid Reference Types
PRO_DTMPLN_THRU	PRO_AXIS, PRO_EDGE, PRO_CURVE, Channel, PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END, PRO_SURFACE (Plane, Cylinder)
PRO_DTMPLN_NORM	PRO_AXIS, PRO_EDGE, PRO_CURVE, Channel PRO_SURFACE (plane)
PRO_DTMPLN_PRL	PRO_SURFACE (plane)
PRO_DTMPLN_OFFS	PRO_SURFACE (plane), PRO_CSYS
PRO_DTMPLN_ANG	PRO_SURFACE (plane)
PRO_DTMPLN_TANG	PRO_SURFACE (cylinder)
PRO_DTMPLN_SEC	PRO_FEATURE (blend)
PRO_DTMPLN_DEF_X	No reference needed
PRO_DTMPLN_DEF_Y	No reference needed
PRO_DTMPLN_DEF_Z	No reference needed
PRO_DTMPLN_THRU_CSYS_XY	PRO_CSYS
PRO_DTMPLN_THRU_CSYS_YZ	PRO_CSYS
PRO_DTMPLN_THRU_CSYS_ZX	PRO_CSYS
PRO_DTMPLN_MIDPLN	Planar reference type: PRO_SURFACE Linear reference types: PRO_AXIS, PRO_EDGE, PRO_CURVE Point reference types: PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END
PRO_DTMPLN_BISECTOR1	Planar reference type: PRO_SURFACE Linear reference types: PRO_AXIS, PRO_EDGE, PRO_CURVE
PRO_DTMPLN_BISECTOR2	Planar reference type: PRO_SURFACE Linear reference types: PRO_AXIS, PRO_EDGE, PRO_CURVE

Note

For constraint type PRO_DTMPLN_TANG, there can be two tangents to a cylindrical surface passing through a single point. Specify a point on the cylindrical surface so that the tangent plane is created through this point or a point nearer to this specified point keeping the tangency condition.

See Creo Parametric online help on datum planes for a detailed description of the valid constraint combinations and references.

Fit Reference Types

The following table describes the corresponding rules for the fit options in detail:

Fit Type	Valid Reference Types
PRO_DTMPLN_FIT_DEFAULT	—
PRO_DTMPLN_FIT_PART	PRO_PART
PRO_DTMPLN_FIT_FEATURE	PRO_FEATURE
PRO_DTMPLN_FIT_SURFACE	PRO_SURFACE
PRO_DTMPLN_FIT_EDGE	PRO_EDGE
PRO_DTMPLN_FIT_AXIS	PRO_AXIS
PRO_DTMPLN_FIT_RADIUS	—
PRO_DTMPLN_FIT_POINT	PRO_POINT

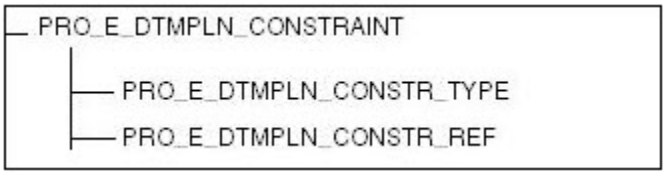
Example 1: Creating a Datum Plane

The sample code in the file UgDatumCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_datum shows how to create a datum plane that is offset from the specified plane. The user selects the reference plane and supplies the offset.

Examples

Example 1: Through a Plane

The element tree structure of a plane through a plane or planar surface is shown in the following figure:



The following table specifies the element tree constraints for this type:

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)

Example 2: Offset to a Plane

The element tree structure of a plane offset to a plane or to a planar surface is shown in the following figure.

```
|--PRO_E_DTMPLN_CONSTRAINT
      |--PRO_E_DTMPLN_CONSTR_TYPE
      |--PRO_E_DTMPLN_CONSTR_REF
      |--PRO_E_DTMPLN_CONSTR_REF_OFFSET
```

The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_OFFS
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)
	PRO_E_DTMPLN_CONSTR_REF_OFFSET	Offset value

Example 3: Offset along a Csys Axis

The element tree structure of a plane offset along the coordinate system axis is shown in the following figure.

```
|--PRO_E_DTMPLN_CONSTRAINT
      |--PRO_E_DTMPLN_CONSTR_TYPE
      |--PRO_E_DTMPLN_CONSTR_REF
      |--PRO_E_DTMPLN_OFF_CSYS
      |--PRO_E_DTMPLN_OFF_CSYS_OFFSET
```

The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_E_DTMPLN_OFF_CSYS_OFFSET
	PRO_E_DTMPLN_CONSTR_REF	PRO_CSYS

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
	PRO_E_DTMPLN_OFF_CSYS	PRO_DTMPLN_OFF_CSYS_X or PRO_DTMPLN_OFF_CSYS_Y or PRO_DTMPLN_OFF_CSYS_Z
	PRO_E_DTMPLN_OFF_CSYS_OFFSET	Offset value

Example 4 : Through a Csys Plane

The element tree structure of a plane passing through a coordinate system plane (XY/YZ/ZX) is shown in the following figure.

```
|--PRO_E_DTMPLN_CONSTRAINT
      |--PRO_E_DTMPLN_CONSTR_TYPE
      |--PRO_E_DTMPLN_CONSTR_REF
```

The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU_CSYS_XY or PRO_DTMPLN_THRU_CSYS_YZ or PRO_DTMPLN_THRU_CSYS_ZX
	PRO_E_DTMPLN_CONSTR_REF	PRO_CSYS

Example 5: Parallel to a Plane and Through a Point

The element tree structure of a plane parallel to a plane or a planar surface and passing through a point is shown in the following figure.

```
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
```

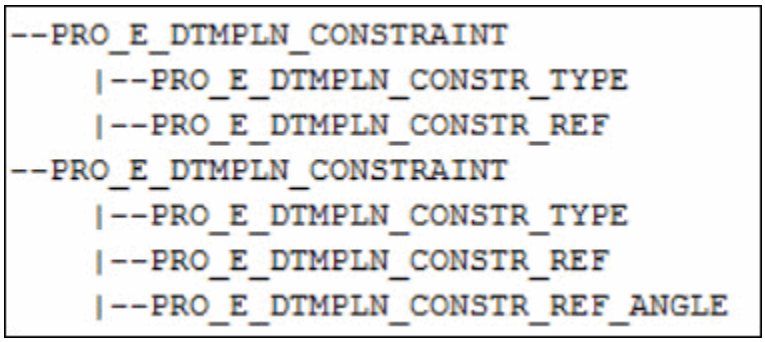
The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_PRL
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT,PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START,PRO_CRV_END

Example 6 : Through an Axis and Angle to a Plane

The element tree structure of a plane passing through an axis and at an angle to a plane is shown in the following figure.



The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_AXIS, PRO_CURVE (STRAIGHT), PRO_EDGE (STRAIGHT)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_ANG
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane),
	PRO_E_DTMPLN_CONSTR_REF_ANGLE	Angle value (-360.0, 360.0)

Example 7: Through a Linear Reference (Axis, Inferred Axis, Straight Edge or Curve) and a Point

The element tree structure of a plane passing through an axis or an inferred axis and a point is shown in the following figure. An inferred axis is the axis of a surface of revolution like, Cylinder, Cone, Sphere, Torus or any other general surface of revolution.

```
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
```

The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_AXIS, PRO_CURVE (STRAIGHT), PRO_EDGE (STRAIGHT, PRO_SURFACE (CYLINDER/CONE/SPHERE/TORUS/General surface of revolution)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END

Example 8: Normal to a Linear Reference (Axis, Inferred Axis, Straight Edge or Curve) and a Point

The element tree structure of a plane normal to an axis or inferred axis and passing through a point is shown in the following figure.

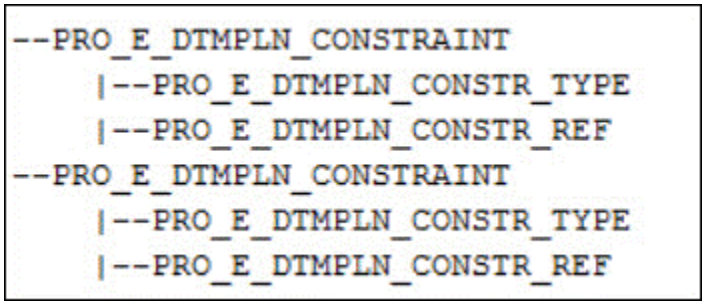
```
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
```

The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_NORM
	PRO_E_DTMPLN_CONSTR_REF	PRO_AXIS, PRO_CURVE (STRAIGHT), PRO_EDGE (STRAIGHT), PRO_SURFACE (CYLINDER/CONE/SPHERE/TORUS/General surface of revolution)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END

Example 9: Midplane to a Plane and Parallel to Another Plane

The element tree structure of a midplane to a plane or a planar surface and parallel to another plane or planar surface, when these two references are parallel, is shown in the following figure.



The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_MIDPLN
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_PRL
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)

Example 10: Midplane to a Plane and at an Angle to Another Plane

The element tree structure of a midplane to a plane or a planar surface and angular to another plane or planar surface, when these two references are intersecting, is shown in the following figure.

```
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
```

The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_ TYPE	PRO_DTMPLN_MIDPLN
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_ TYPE	PRO_DTMPLN_BISECTOR1 PRO_DTMPLN_BISECTOR2
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)

Example 11: Midplane to a Plane and Midplane to a Point

The element tree structure of a midplane to a plane or a planar surface and midplane to a point, where the point does not lie on the plane, is shown in the following figure.

```
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
--PRO_E_DTMPLN_CONSTRAINT
    |--PRO_E_DTMPLN_CONSTR_TYPE
    |--PRO_E_DTMPLN_CONSTR_REF
```

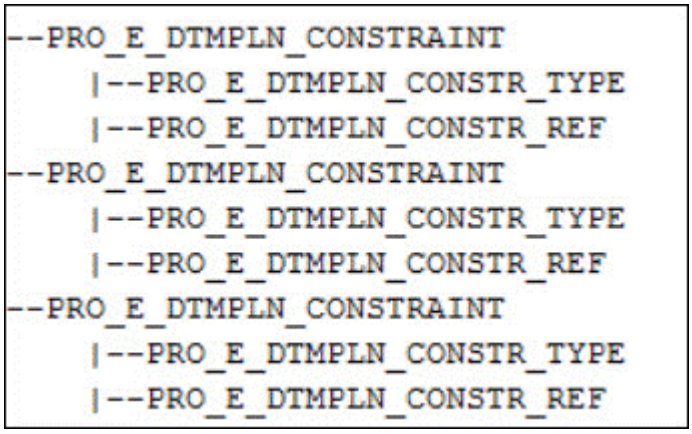
The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_ TYPE	PRO_DTMPLN_MIDPLN
	PRO_E_DTMPLN_CONSTR_REF	PRO_SURFACE (Plane)
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_ TYPE	PRO_DTMPLN_MIDPLN

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END

Example 12: Through Three Points

The element tree structure of a plane passing through three non-collinear points is shown in the following figure.



The following table specifies the element tree constraints for this type.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMPLN_CONSTRAINT (Constraint 1)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START,PRO_CRV_END
PRO_E_DTMPLN_CONSTRAINT (Constraint 2)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END
PRO_E_DTMPLN_CONSTRAINT (Constraint 3)	PRO_E_DTMPLN_CONSTR_TYPE	PRO_DTMPLN_THRU
	PRO_E_DTMPLN_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END

Example 13: To Flip Direction

The datum plane normal that is the Z-direction can be flipped for any of the types. The following table specifies the element tree to flip the plane.

Element ID	Valid Value
PRO_E_DTMPLN_FLIP_DIR	PRO_DTMPLN_FLIP_DIR_NO or PRO_DTMPLN_FLIP_DIR_YES

Example 14: To Fit Outline to a Reference

The datum plane outline can be fit to a reference. The following figure shows the element tree structure using the fit elements.

```
|--PRO_E_DTMPLN_FIT
      |--PRO_E_DTMPLN_FIT_TYPE
      |--PRO_E_DTMPLN_FIT_REF
```

The following table specifies the element tree to use fit elements.

Fit Compound Element	Fit Member Elements	Valid Value
PRO_E_DTMPLN_FIT	PRO_E_DTMPLN_FIT _TYPE	PRO_DTMPLN_FIT_PART
	PRO_E_DTMPLN_FIT _REF	PRO_PART

Datum Point Features

The element tree for a datum point feature is documented in the header file `ProDtmPnt.h`. Apart from the usual elements for the tree root and feature type, a datum point contains the datum point type. The types of datum points available are:

- Sketched Datum Point on page 817
- Field Datum Point on page 818
- Offset Csys Datum Point on page 819
- General Datum Point on page 820

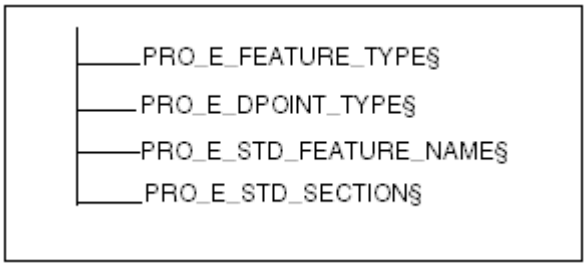
Sketched Datum Point

A sketched datum point is created by sketching the point in the sketcher mode after specifying the plane on which the user wants to create a point.

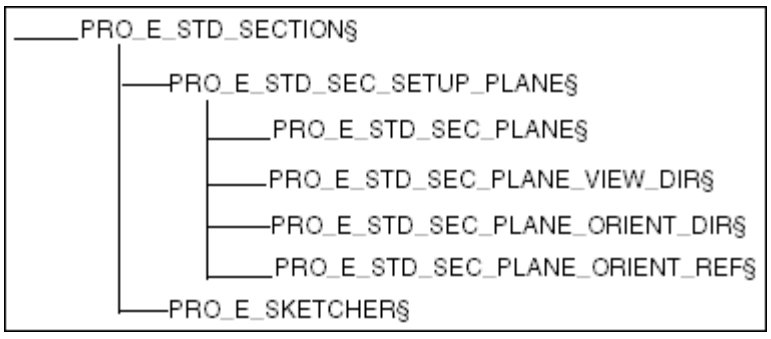
Note

The sketched datum point is obsolete. Hence, the element tree for sketched datum point defined in the header file `ProDtmPnt.h` is no longer supported. The sketched datum points are consolidated within the sketched feature as a geometry point. The geometry entities within the sketched feature generate the corresponding datum entities. To create new sketched datum points, you must use the element tree for sketched datum curves defined in the header file `ProDtmCrv.h`. Refer to the section Sketched Datum Curves on page 848, for more information on sketched datum curves.

The following figure shows the element tree for a sketched datum point.



Define the following sub elements of `PRO_E_STD_SECTION` to complete the sketched datum point feature.



See the chapter Element Trees: Sketched Features on page 1004 for techniques that must be used to create a sketched feature, like a sketched datum point.

Feature Elements

The following table describes the elements of the element tree for sketched datum points:

Element ID	Element Name	Data Type	Valid Value
PRO_E_FEATURE_TYPE	Feature Type	PRO_VALUE_TYPE_INT	PRO_FEAT_DATUM_POINT
PRO_E_DPOINT_TYPE	Datum Point Type	PRO_VALUE_TYPE_INT	PRO_DPOINT_TYPE_SKETCHED
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_STD_SECTION	Section	Compound	See ProStdSection.h

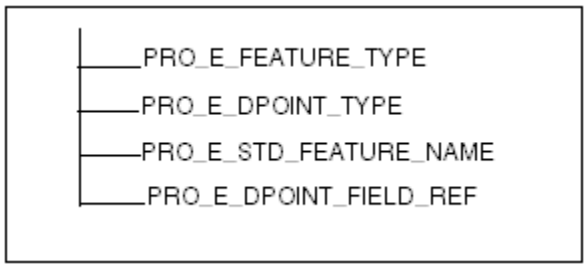
Example 2: Creating a Sketched Datum Point

The sample code in the file `UgSketchedPointCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a Sketched Datum Point. The user is prompted to select the sketching planes, orientation planes, and then the reference edges for the sketch. The user is also required to enter the X and Y offsets to be applied to the sketch from the projected edges.

Field Datum Point

A field datum point is created by selecting any point on a surface, edge, curve, or quilt. The point is located depending on the UV parameters. The location of the field point depends on the UV values of the point on the surface, edge, curve, or quilt.

The following figure shows the element tree for the field datum point.



Feature Elements

The following table describes the elements of the element tree for the field datum points:

Element ID	Element Name	Data Type	Valid Value
PRO_E_FEATURE_TYPE	Feature Type	PRO_VALUE_TYPE_INT	PRO_FEAT_DATUM_POINT
PRO_E_DPOINT_TYPE	Datum Point Type	PRO_VALUE_TYPE_INT	PRO_DPOINT_TYPE_FIELD

Element ID	Element Name	Data Type	Valid Value
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_DPOINT_FIELD_REF	Placement reference	PRO_VALUE_TYPE_SELECTION	Surface, Edge, Curve, or Quilt. Note: UV is used to specify exact location.

Example 3: Creating a Field Datum Point

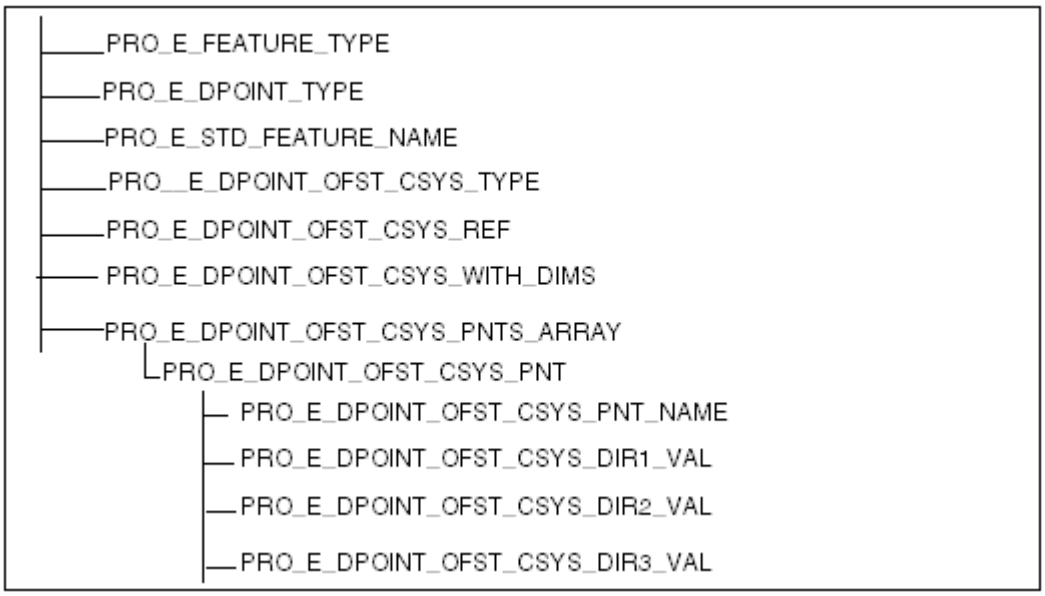
The sample code in the file `UgFieldPointCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a Field Datum Point on an edge or surface. The user is prompted to select a point on a curve or a surface.

Offset Csys Datum Point

An Offset Csys Datum point is created using the coordinate system and values along the coordinate axes. Three types of coordinate systems can be used:

- Cartesian—Requires values along X, Y, Z axis.
- Cylindrical—Requires values along R, theta, Z axis.
- Spherical—Requires values along r,phi theta axis.

The following figure shows the element tree for Offset Csys Datum Point.



Feature Elements

The following table describes the elements in the element tree for datum points.

Element ID	Element Name	Data Type	Valid Value
PRO_E_FEATURE_TYPE	Feature Type	PRO_VALUE_TYPE_INT	PRO_FEAT_DATUM_POINT
PRO_E_DPOINT_TYPE	Datum Point Type	PRO_VALUE_TYPE_INT	PRO_DPOINT_TYPE_OFFSET_CSYS
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_DPOINT_OFST_CSYS_TYPE	Reference Csys Type	PRO_VALUE_TYPE_INT	See ProDtmpntOffCsysType
PRO_E_DPOINT_OFST_CSYS_REF	Reference Csys	PRO_VALUE_TYPE_SELECTION	Csys
PRO_E_DPOINT_OFST_CSYS_WITH_DIMS	Parametric or Explicit with or without dimensions	PRO_VALUE_TYPE_INT	PRO_B_TRUE or PRO_B_FALSE
PRO_E_DPOINT_OFST_CSYS_PNTS_ARRAY	Array of Points List		
PRO_E_DPOINT_OFST_CSYS_PNT	One Point	Compound	
PRO_E_DPOINT_OFST_CSYS_PNT_NAME	Point Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_DPOINT_OFST_CSYS_DIR1_VAL	X, R, or pi	PRO_VALUE_TYPE_DOUBLE	Depends on PRO_E_DPOINT_OFST_CSYS_TYPE
PRO_E_DPOINT_OFST_CSYS_DIR2_VAL	Y, theta, or phi	PRO_VALUE_TYPE_DOUBLE	Depends on PRO_E_DPOINT_OFST_CSYS_TYPE
PRO_E_DPOINT_OFST_CSYS_DIR3_VAL	Z, Z, or theta	PRO_VALUE_TYPE_DOUBLE	Depends on PRO_E_DPOINT_OFST_CSYS_TYPE

Example 4: Creating an Offset Csys Datum Point

The sample code in the file UgOffsetPointCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create an Offset Datum Point at an offset from a specified coordinate system. The user is prompted to select the coordinate system.

General Datum Point

A general datum point is created and constrained based on the selection context. The supported types are:

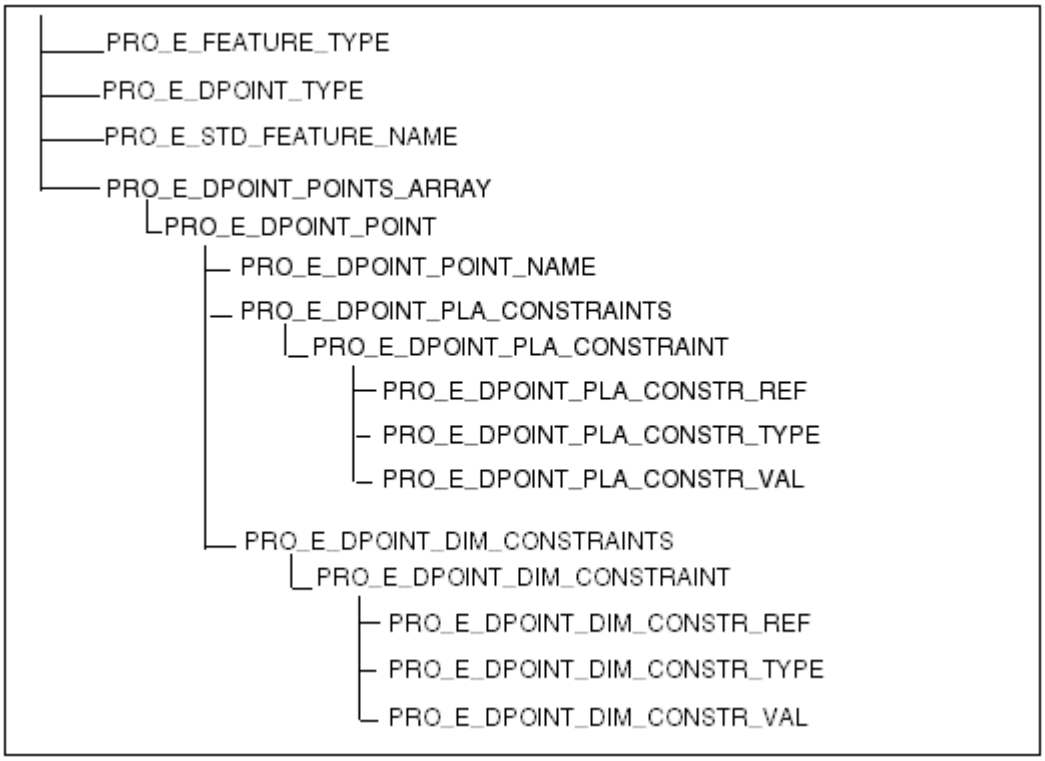
- Point on vertex
- Offset point
- Project point
- Point at intersection of three surfaces

- On or Offset surface
- Point at intersection of curve and surface
- Center of curve or surface
- Point at intersection of two curves
- Point on curve

When there are multiple intersections, the point location of general datum point depends on the following:

- Point at intersection of edge and edge—t value of point on second edge
- Point at intersection of edge and plane—t value of point on edge
- Point at intersection of curve and plane—t value of point on curve
- Point at intersection of two curves—t value of point on second curve
- Point at intersection of curve and surface—t value of point on curve
- Point at intersection of curve and axis—t value of point on curve

The following figure shows the element tree of a general datum point.



Feature Elements

The following table describes the elements of the element tree for datum points.

Element ID	Element Name	Data Type	Valid Value
PRO_E_FEATURE_TYPE	Feature Type	PRO_VALUE_TYPE_INT	PRO_FEAT_DATUM_POINT
PRO_E_DPOINT_TYPE	Datum Point Type	PRO_VALUE_TYPE_INT	PRO_DPOINT_TYPE_GENERAL
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_DPOINT_POINTS_ARRAY	Points List	Array	Not applicable
PRO_E_DPOINT_POINT	One Point	Compound	Not applicable
PRO_E_DPOINT_POINT_NAME	Point Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_DPOINT_PLA_CONSTRAINTS	Placement Constraints	Array	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT	One Placement Constraint	Compound	Not applicable
PRO_E_DPOINT_PLA_CONSTR_REF	Placement Reference	PRO_VALUE_TYPE_SELECTION	Depends on the context. See PRO_E_DPOINT_PLA_CONSTR_REF.
PRO_E_DPOINT_PLA_CONSTR_TYPE	Constraint Type	PRO_VALUE_TYPE_INT	See ProDtmpntConstrType.
PRO_E_DPOINT_PLA_CONSTR_VAL	Value	PRO_VALUE_TYPE_DOUBLE	
PRO_E_DPOINT_DIM_CONSTRAINTS	Dimension Constraints	Array	Not applicable
PRO_E_DPOINT_DIM_CONSTRAINT	One Dimension Constraint	Compound	Not applicable
PRO_E_DPOINT_DIM_CONSTR_REF	Dimension Reference	PRO_VALUE_TYPE_SELECTION	See Placement Constraint References on page 822.
PRO_E_DPOINT_DIM_CONSTR_TYPE	Constraint Type	PRO_VALUE_TYPE_INT	Depends on the context. See Constraint Type on page 823.
PRO_E_DPOINT_DIM_CONSTR_VAL	Value	PRO_VALUE_TYPE_DOUBLE	See ProDtmpntConstrType

Placement Constraint References

Valid values for the PRO_E_DPOINT_PLA_CONSTR_REF placement reference are as follows:

- Curve—SEL_3D_CURVE, SEL_3D_CABLE, SEL_IGES_WF
- Edge—SEL_3D_EDG
- Axis—SEL_3D_AXIS
- Vertex—SEL_3D_VERT or SEL_CURVE_END

- CSYS—SEL_3D_CSYS
- Surface—SEL_3D_SRF, SEL_3D_SRF_LIST
- Datum Pnt—SEL_3D_PNT

Placement Constraint Type

Valid values for PRO_E_DPOINT_PLA_CONSTR_TYPE are as follows:

- PRO_DTMPNT_CONSTR_TYPE_ON
- PRO_DTMPNT_CONSTR_TYPE_OFFSET
- PRO_DTMPNT_CONSTR_TYPE_CENTER
- PRO_DTMPNT_CONSTR_TYPE_PARALLEL
- PRO_DTMPNT_CONSTR_TYPE_NORMAL
- PRO_DTMPNT_CONSTR_TYPE_PROJECT
- PRO_DTMPNT_CONSTR_TYPE_CARTESIAN
- PRO_DTMPNT_CONSTR_TYPE_CYLINDRICAL
- PRO_DTMPNT_CONSTR_TYPE_SPHERICAL

The last three are used when defining a point offset to a coordinate system.

Constraint References

Valid values for the PRO_E_DPOINT_DIM_CONSTR_REF dimension references are as following:

- Curve—SEL_3D_CURVE, SEL_3D_CABLE, SEL_CRV_PNT, SEL_IGES_WF
- Edge—SEL_3D_EDG, SEL_EDG_PNT
- Axis—SEL_3D_AXIS
- Coordinate system—SEL_3D_CSYS
- Vertex—SEL_3D_VERT or SEL_CURVE_END
- Surface—SEL_3D_SRF, SEL_SRF_PNT, SEL_3D_SRF_LIST
- Coordinate system axis—SEL_3D_CSYS_AXIS
- Datum Point—SEL_3D_PNT

Constraint Type

Valid values for PRO_E_DPOINT_DIM_CONSTR_TYPE are as follows:

- PRO_DTMPNT_CONSTR_TYPE_OFFSET
- PRO_DTMPNT_CONSTR_TYPE_LENGTH
- PRO_DTMPNT_CONSTR_TYPE_RATIO

- PRO_DTMPNT_CONSTR_TYPE_LENGTH_END
- PRO_DTMPNT_CONSTR_TYPE_RATIO
- PRO_DTMPNT_CONSTR_TYPE_RATIO_END
- PRO_DTMPNT_CONSTR_TYPE_ALONG_X
- PRO_DTMPNT_CONSTR_TYPE_ALONG_Y
- PRO_DTMPNT_CONSTR_TYPE_ALONG_Z

Example 5: Creating General Datum Point

The sample code in the file UgGeneralPointCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a General Datum Point formed at the intersection of three selected surfaces. The user is prompted to select the three surfaces.

Examples

Example 1: Point on a Vertex

To create a datum point on the vertex, the following constraints are required.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Vertex
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

Example 2: Offset Point

To create one or more datum points at an offset, the following constraints are required.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Vertex, Csys, or DPnt
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_OFFSET
	PRO_E_DPOINT_PLA_CONSTR_VAL	Offset value.

The following tables provide valid values for Constraint 2. You can create a point at an offset using values from one of the following tables for Constraint 2.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Curve, Edge or Axis
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_PARALLEL
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

OR

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_NORMAL
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

OR

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Csys Axis
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_PARALLEL
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

OR

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Csys
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_CARTESIAN or PRO_DTMPNT_CONSTR_TYPE_CYLINDRICAL or PRO_DTMPNT_CONSTR_TYPE_SPHERICAL
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

The following table provides valid values for dimension constraints.

Dimension Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_DIM_CONSTRAINT (Constraint 3)	PRO_E_DPOINT_DIM_CONSTR_REF	Not applicable
	PRO_E_DPOINT_DIM_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ALONG_X

Dimension Constraint Element	Reference Element	Valid Value
	PRO_E_DPOINT_DIM_CONSTR_VAL	Offset Value
PRO_E_DPOINT_DIM_CONSTRAINT (Constraint 4)	PRO_E_DPOINT_DIM_CONSTR_REF	Not applicable
	PRO_E_DPOINT_DIM_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ALONG_Y
	PRO_E_DPOINT_DIM_CONSTR_VAL	Offset Value
PRO_E_DPOINT_DIM_CONSTRAINT (Constraint 5)	PRO_E_DPOINT_DIM_CONSTR_REF	Not applicable
	PRO_E_DPOINT_DIM_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ALONG_Z
	PRO_E_DPOINT_DIM_CONSTR_VAL	Offset Value

As per Offset types the values are as follows:

	CARTESIAN	CYLINDRICAL	SPHERICAL
PRO_DTMPNT_CONSTR_TYPE_ALONG_X	X	R	RHO
PRO_DTMPNT_CONSTR_TYPE_ALONG_Y	Y	THETA	PHI
PRO_DTMPNT_CONSTR_TYPE_ALONG_Z	Z	Z	THETA

Example 3: Point at Intersection of Three Surfaces

To create a datum point at the intersection of three surfaces, use the following constraints. Each surface can be a part surface, surface feature, or datum plane.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 3)	PRO_E_DPOINT_PLA_CONSTR_REF	Surface

Placement Constraint Element	Reference Element	Valid Value
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

Example 4: Point On a Surface or Offset from a Surface

The following constraints are required to create a point on a surface or at an offset distance from a surface:

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON or PRO_DTMPNT_CONSTR_TYPE_OFFSET
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Edge or Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_OFFSET
	PRO_E_DPOINT_PLA_CONSTR_VAL	Offset value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 3)	PRO_E_DPOINT_PLA_CONSTR_REF	Edge or Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_OFFSET
	PRO_E_DPOINT_PLA_CONSTR_VAL	Offset value

Example 5: Point at Intersection of a Curve and a Surface

To create a datum point at the intersection of a curve and a surface, use the following constraints. The curve can be a part edge, surface feature edge, datum curve, axis, or an imported datum curve. The surface can be a part surface, surface feature, or datum plane.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Curve, axis, edge, or surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	- If value of constraint 1 is Curve, Axis, or Edge, the value of

Placement Constraint Element	Reference Element	Valid Value
		constraint 2 is surface. - If value of constraint 1 is surface, the value of constraint 2 is Curve, Axis, or Edge.
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

Note

If more than one intersections exist, the point is created at the intersection nearest to the curve reference parameter value.

Example 6: Point At Center of Curve or Surface

To create a datum point at the center of an arc or circle entity, use the following constraints.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Curve, edge, or surface (Sphere)
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_CENTER
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

Example 7: Point at Intersection of Two Curves

To create a point at intersection of two curves, use the following constraints.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Curve, edge, or axis
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Curve, edge, or axis
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

Note

If more than one intersections exist, the point is created at the intersection nearest to the second reference parameter value.

Example 8: Point On Curve

To create a datum point on a curve, the following constraints are required.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Curve, edge, or axis (It is valid with offset plane)
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

The following tables provide valid values for constraint 2. You can create a point on curve using values from one of the following tables for constraint 2.

Use the following values for constraint 2 if the length of curve from the start point or the end point is used to locate the point.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_DIM_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_DIM_CONSTR_REF	Curve (Use the same curve as used in Constraint 1)
	PRO_E_DPOINT_DIM_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_LENGTH or PRO_DTMPNT_CONSTR_TYPE_LENGTH_END
	PRO_E_DPOINT_DIM_CONSTR_VAL	Length value (from curve start point or end point)

Use the following values for constraint 2 if the ratio of distance from the start point or the end point is used to locate the point.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_DIM_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_DIM_CONSTR_REF	Curve (Use the same curve as used in contrarian 1)
	PRO_E_DPOINT_DIM_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_RATIO or PRO_DTMPNT_CONSTR_TYPE_RATIO_END
	PRO_E_DPOINT_DIM_CONSTR_VAL	Ratio value (from curve start or end)

Use the following values for constraint 2 if the offset surface is used to locate the point on curve.

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_DIM_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_DIM_CONSTR_REF	Surface
	PRO_E_DPOINT_DIM_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_OFFSET
	PRO_E_DPOINT_DIM_CONSTR_VAL	Offset value

Example 9: Project Datum Point On a Planar surface, Datum Plane, Datum Axis, Linear Curve or Linear Edge

To project a datum point on a planar surface, datum plane, or datum axis, the following constraints are required.

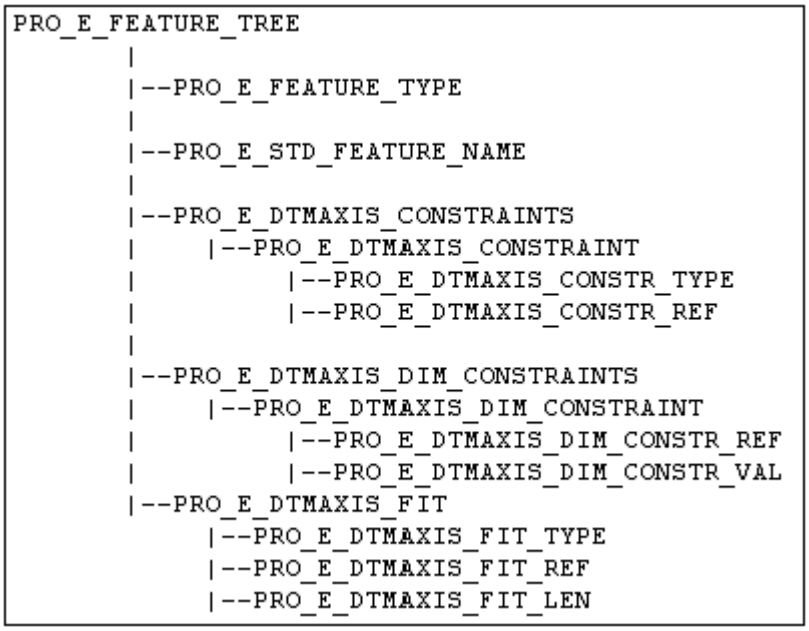
Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 1)	PRO_E_DPOINT_PLA_CONSTR_REF	Datum point, end of curve, or vertex
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_PROJECT
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Axis, curve, or edge
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

OR

Placement Constraint Element	Reference Element	Valid Value
PRO_E_DPOINT_PLA_CONSTRAINT (Constraint 2)	PRO_E_DPOINT_PLA_CONSTR_REF	Surface
	PRO_E_DPOINT_PLA_CONSTR_TYPE	PRO_DTMPNT_CONSTR_TYPE_ON
	PRO_E_DPOINT_PLA_CONSTR_VAL	Not applicable

Datum Axis Features

The basic element tree for creating axes is available in the include file `ProDtmAxis.h`. The following figure shows the basic structure of the element tree.



Creo Parametric TOOLKIT supports creation of the following types of datum axes:

- Point on Surface on page 832
- Tangent on page 834
- Example 3: Normal to a Linear Reference (Axis, Inferred Axis, Straight Edge or Curve) on page 833
- Example 4: Parallel to a Linear Reference (Axis, Inferred Axis, Straight Edge or Curve) on page 834
- Example 5: Through Edge or Surface on page 835
- Two Planes on page 836
- Two Points on page 836
- Normal Planes on page 837

There is no single element that indicates the type in constraints element tree. The type is determined implicitly based on the constraint type and references. The types of the datum axis constraints for the references are defined by the enumerated type `ProDtmaxisConstrType` and are as follows:

- `PRO_DTMAXIS_CONSTR_TYPE_NORMAL`— Positions the datum axis normal to the selected reference.
- `PRO_DTMAXIS_CONSTR_TYPE_THRU`— Positions the datum axis through the selected reference.
- `PRO_DTMAXIS_CONSTR_TYPE_TANGENT`— Positions the datum axis tangent to the selected reference.

- PRO_DTMAXIS_CONSTR_TYPE_CENTER— Positions the datum axis through the center of the selected planar circular edge or curve and normal to the plane on which the selected curve or edge lies.
- PRO_DTMAXIS_CONSTR_TYPE_PARALLEL— Positions the datum axis parallel to the selected reference.

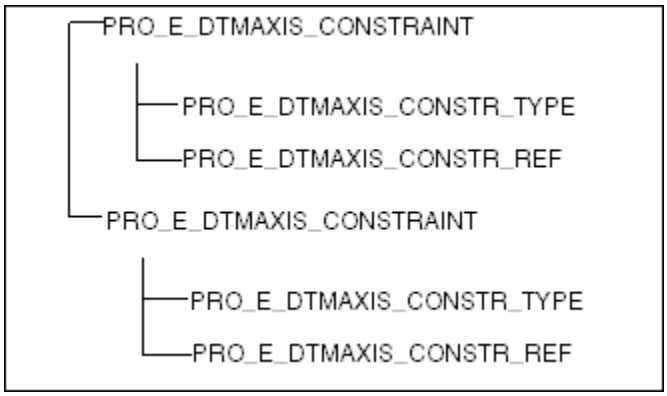
Example 6: Creating a Datum Axis

The sample code in the file UgDatumAxisCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a datum axis at the intersection of two selected surfaces. The user is prompted to select the two surfaces.

Examples

Example 1: Point on Surface

The element tree structure of the axis, created with type as PRO_DTMAXIS_PNT_SURF, is shown in the following figure.

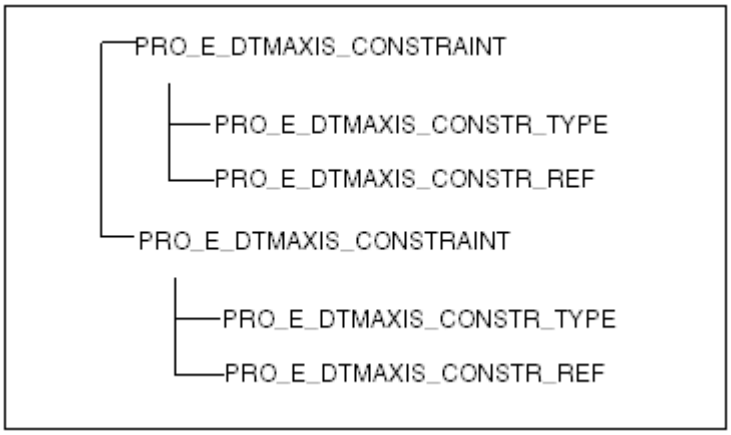


The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the point on surface type of axis.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_POINT
PRO_E_DTMAXIS_CONSTRAINT (Constraint 2)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_NORMAL
	PRO_E_DTMAXIS_CONSTR_REF	PRO_SURFACE

Example 2: Tangent

The element tree structure of the axis, created with type as Tangent, is shown in the following figure.

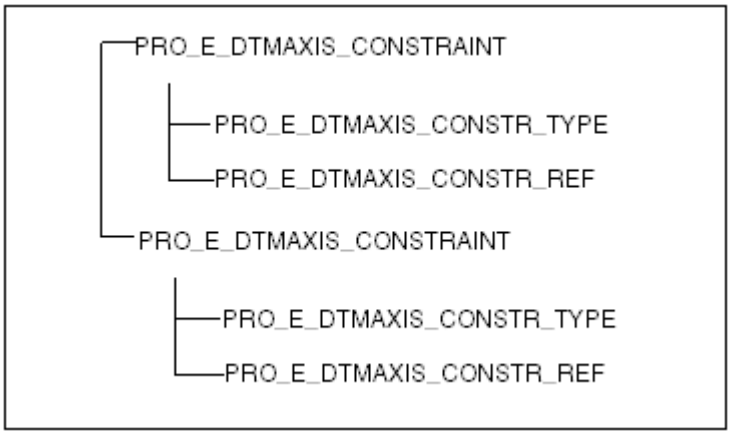


The following table specifies the constraints for the `PRO_E_DTMAXIS_CONSTRAINT` elements in the element tree for the tangent type of axis.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
`PRO_E_DTMAXIS_CONSTRAINT` (Constraint 1)	`PRO_E_DTMAXIS_CONSTR_TYPE`	`PRO_DTMAXIS_CONSTR_TYPE_TANGENT`
	`PRO_E_DTMAXIS_CONSTR_REF`	`PRO_EDGE`, `PRO_CURVE`
`PRO_E_DTMAXIS_CONSTRAINT` (Constraint 2)	`PRO_E_DTMAXIS_CONSTR_TYPE`	`PRO_DTMAXIS_CONSTR_TYPE_THRU`
	`PRO_E_DTMAXIS_CONSTR_REF`	`PRO_POINT`, `PRO_EDGE_START`, `PRO_EDGE_END`,`PRO_CRV_START`, `PRO_CRV_END`

Example 3: Normal to a Linear Reference (Axis, Inferred Axis, Straight Edge or Curve)

The element tree structure of the axis, created with type as Normal, is shown in the following figure.



The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the Normal type of axis.

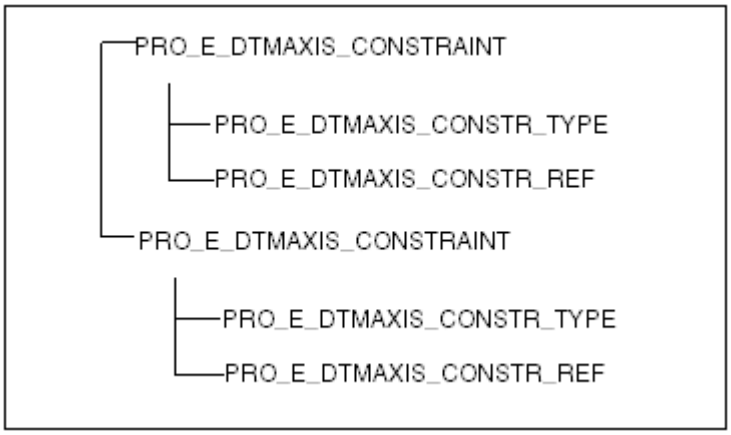
Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END
PRO_E_DTMAXIS_CONSTRAINT (Constraint 2)	PRO_E_DTMAXIS_CONSTR_TYPE	
	PRO_E_DTMAXIS_CONSTR_REF	PRO_EDGE (STRAIGHT), PRO_CURVE (STRAIGHT), PRO_AXIS, PRO_SURFACE (CYLINDER/CONE/SPHERE/TORUS/General surface of revolution). The inferred axis will be used as the reference.

Note

For the Normal type of datum axis creation, the reference provided for the first constraint that is, PRO_DTMAXIS_CONSTR_TYPE_THRU should not lie on the reference provided for second constraint that is PRO_DTMAXIS_CONSTR_TYPE_NORMAL.

Example 4: Parallel to a Linear Reference (Axis, Inferred Axis, Straight Edge or Curve)

The element tree structure of the axis, created with type as Parallel, is shown in the following figure.

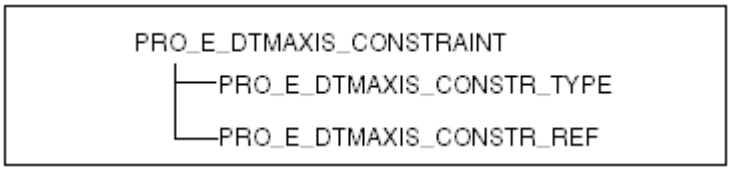


The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the Parallel type of axis.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END
PRO_E_DTMAXIS_CONSTRAINT (Constraint 2)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_PARALLEL
	PRO_E_DTMAXIS_CONSTR_REF	PRO_EDGE (STRAIGHT), PRO_CURVE (STRAIGHT), PRO_AXIS, PRO_SURFACE (CYLINDER/CONE/SPHERE/TORUS/ General surface of revolution). The inferred axis will be used as reference.

Example 5: Through Edge or Surface

The element tree structure of the axis, created with type Through an Edge or a Surface, is shown in the following figure.



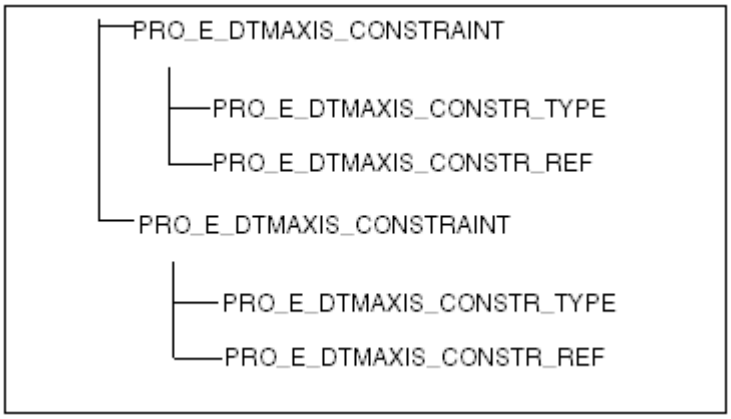
The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the through type of axis.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
	PRO_E_DTMAXIS_CONSTR_REF	PRO_EDGE (Straight), PRO_SURFACE (Cylinder)

Example 6: Two Planes

The element tree structure of the axis, created using the type as two planes, is as shown in the following figure.

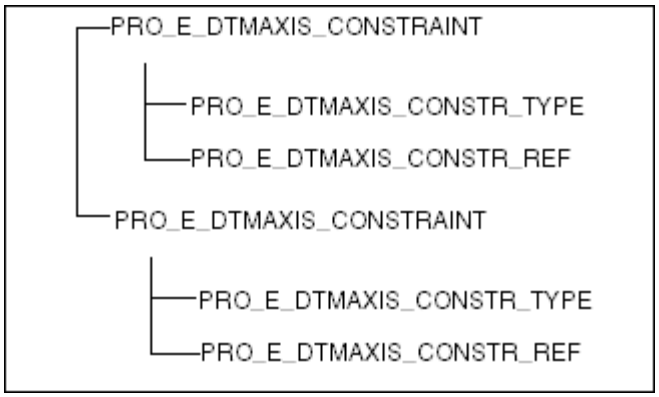


The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the two planes reference scheme.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_SURFACE (Planar)
PRO_E_DTMAXIS_CONSTRAINT (Constraint 2)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_SURFACE (Planar)

Example 7: Two Points

The element tree structure of the axis, created using the type as two points, is shown in the following figure.

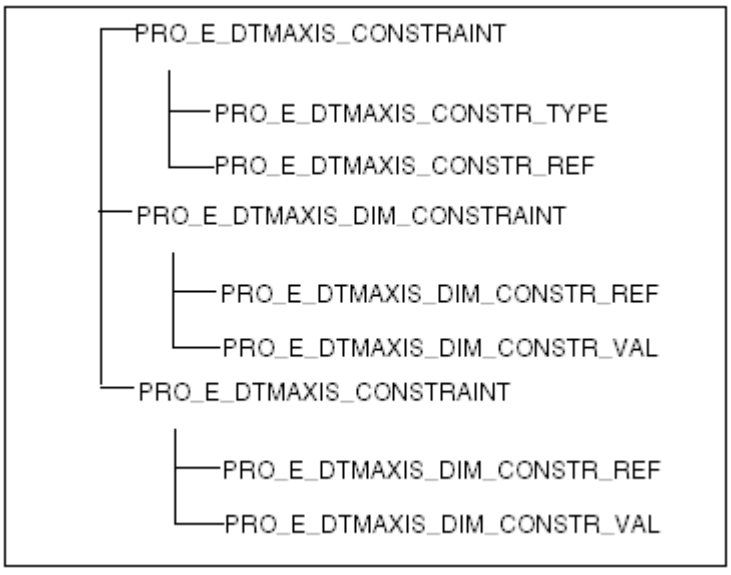


The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the two points type of axis.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END
PRO_E_DTMAXIS_CONSTRAINT (Constraint 2)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_THRU
	PRO_E_DTMAXIS_CONSTR_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END

Example 8: Normal Planes

The element tree structure of the axis, created using the type as Normal Planes, is shown in the following figure.

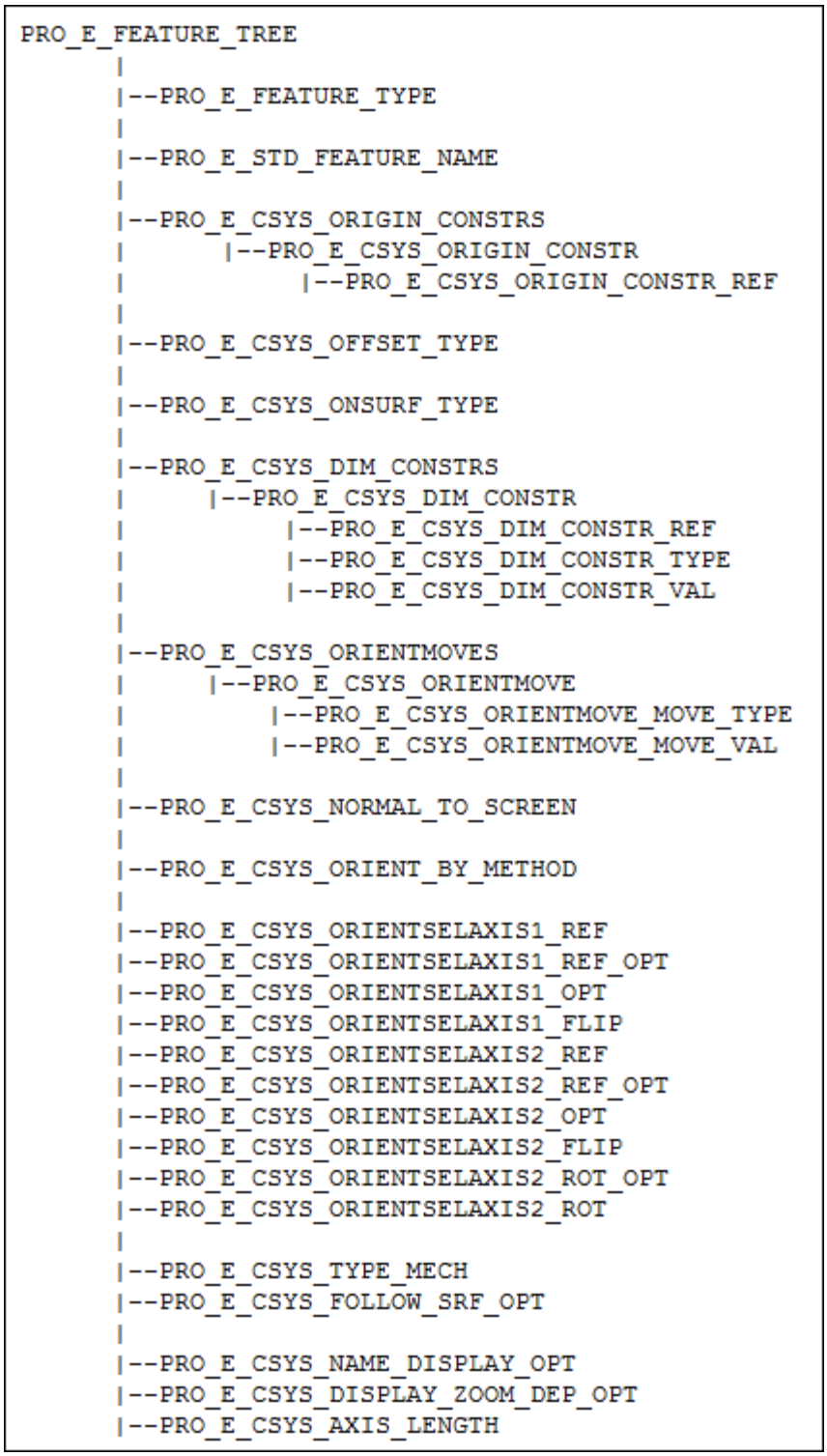


The following table specifies the constraints for the PRO_E_DTMAXIS_CONSTRAINT elements in the element tree for the normal plane type of axis.

Placement Constraint Element	Placement Constraint Member Elements	Valid Value
PRO_E_DTMAXIS_CONSTRAINT (Constraint 1)	PRO_E_DTMAXIS_CONSTR_TYPE	PRO_DTMAXIS_CONSTR_TYPE_NORMAL
	PRO_E_DTMAXIS_CONSTR_REF	PRO_SURFACE (Planar)
PRO_E_DTMAXIS_DIM_CONSTRAINT (Constraint 2)	PRO_E_DTMAXIS_DIM_CONSTR_REF	PRO_SURFACE (Planar), PRO_AXIS, PRO_EDGE
	PRO_E_DTMAXIS_DIM_CONSTR_VAL	Valid dimension
PRO_E_DTMAXIS_DIM_CONSTRAINT (Constraint 3)	PRO_E_DTMAXIS_DIM_CONSTR_REF	PRO_SURFACE (Planar), PRO_AXIS, PRO_EDGE
	PRO_E_DTMAXIS_DIM_CONSTR_VAL	Valid dimension

Datum Coordinate System Features

The following figure illustrates the general structure of the element tree for coordinate system features.


PRO_E_FEATURE_TREE
|--PRO_E_FEATURE_TYPE
|--PRO_E_STD_FEATURE_NAME
|--PRO_E_CSYS_ORIGIN_CONSTRS
|--PRO_E_CSYS_ORIGIN_CONSTR
|--PRO_E_CSYS_ORIGIN_CONSTR_REF
|--PRO_E_CSYS_OFFSET_TYPE
|--PRO_E_CSYS_ONSURF_TYPE
|--PRO_E_CSYS_DIM_CONSTRS
|--PRO_E_CSYS_DIM_CONSTR
|--PRO_E_CSYS_DIM_CONSTR_REF
|--PRO_E_CSYS_DIM_CONSTR_TYPE
|--PRO_E_CSYS_DIM_CONSTR_VAL
|--PRO_E_CSYS_ORIENTMOVES
|--PRO_E_CSYS_ORIENTMOVE
|--PRO_E_CSYS_ORIENTMOVE_MOVE_TYPE
|--PRO_E_CSYS_ORIENTMOVE_MOVE_VAL
|--PRO_E_CSYS_NORMAL_TO_SCREEN
|--PRO_E_CSYS_ORIENT_BY_METHOD
|--PRO_E_CSYS_ORIENTSELAXIS1_REF
|--PRO_E_CSYS_ORIENTSELAXIS1_REF_OPT
|--PRO_E_CSYS_ORIENTSELAXIS1_OPT
|--PRO_E_CSYS_ORIENTSELAXIS1_FLIP
|--PRO_E_CSYS_ORIENTSELAXIS2_REF
|--PRO_E_CSYS_ORIENTSELAXIS2_REF_OPT
|--PRO_E_CSYS_ORIENTSELAXIS2_OPT
|--PRO_E_CSYS_ORIENTSELAXIS2_FLIP
|--PRO_E_CSYS_ORIENTSELAXIS2_ROT_OPT
|--PRO_E_CSYS_ORIENTSELAXIS2_ROT
|--PRO_E_CSYS_TYPE_MECH
|--PRO_E_CSYS_FOLLOW_SRF_OPT
|--PRO_E_CSYS_NAME_DISPLAY_OPT
|--PRO_E_CSYS_DISPLAY_ZOOM_DEP_OPT
|--PRO_E_CSYS_AXIS_LENGTH

Feature Elements

The following table describes the elements in the element tree for coordinate system feature.

Element Id	Element Name	Data Type	Valid Values
PRO_E_FEATURE_TYPE	Feature Type	PRO_VALUE_TYPE_INT	PRO_FEAT_CSYS
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING	
PRO_E_CSYS_ORIGIN_CONSTRS	Origin Constraints	Array	
PRO_E_CSYS_ORIGIN_CONSTR	Origin Constraint	Compound	
PRO_E_CSYS_ORIGIN_CONSTR_REF	Origin Reference	PRO_VALUE_TYPE_SELECTION	
PRO_E_CSYS_OFFSET_TYPE	Origin Offset Type	PRO_VALUE_TYPE_INT	ProCsysOffsetType
PRO_E_CSYS_ONSURF_TYPE	On Surface Type	PRO_VALUE_TYPE_INT	ProCsysOnSurfType
PRO_E_CSYS_DIM_CONSTRS	Dimension Constraints	Array	
PRO_E_CSYS_DIM_CONSTR	Dimension Constraint	Compound	
PRO_E_CSYS_DIM_CONSTR_REF	Dimension Constraint Reference	PRO_VALUE_TYPE_SELECTION	
PRO_E_CSYS_DIM_CONSTR_TYPE	Dimension Constraint Type	PRO_VALUE_TYPE_INT	ProCsysDimConstrType
PRO_E_CSYS_DIM_CONSTR_VAL	Dimension Constraint Value	PRO_VALUE_TYPE_DOUBLE	
PRO_E_CSYS_ORIENTMOVES	Orientation Moves	Array	
PRO_E_CSYS_ORIENTMOVE		Compound	
PRO_E_CSYS_ORIENTMOVE_MOVE_TYPE	Move Type	PRO_VALUE_TYPE_INT	ProCsysOrientMoveMoveOpt
PRO_E_CSYS_ORIENTMOVE_MOVE_VAL	Move Value	PRO_VALUE_TYPE_DOUBLE	
PRO_E_CSYS_NORMAL_TO_SCREEN	Set Z Normal To Screen	PRO_VALUE_TYPE_INT	ProCsysOrientMovesNrmScrnOpt
PRO_E_CSYS_ORIENT_BY_METHOD	Orient By Method	PRO_VALUE_TYPE_INT	ProCsysOrientByMethod
PRO_E_CSYS_ORIENTSELAXIS1_REF	First Axis Reference	PRO_VALUE_TYPE_SELECTION	
PRO_E_CSYS_ORIENTSELAXIS1_REF_OPT	First Axis Reference Option	PRO_VALUE_TYPE_INT	ProCsysDirCsysRefOpt
PRO_E_CSYS_ORIENTSELAXIS1_	First Axis Option	PRO_VALUE_TYPE_INT	ProCsysOrientMoveAxisOpt

Element Id	Element Name	Data Type	Valid Values
OPT			
PRO_E_CSYS_ORIENTSELAXIS1_FLIP	Flip first direction	-	
PRO_E_CSYS_ORIENTSELAXIS2_REF	Second Axis Reference	PRO_VALUE_TYPE_SELECTION	
PRO_E_CSYS_ORIENTSELAXIS2_REF_OPT	Second Axis Reference Option	PRO_VALUE_TYPE_INT	ProCsysDirCsysRefOpt
PRO_E_CSYS_ORIENTSELAXIS2_OPT	Second Axis Option	PRO_VALUE_TYPE_INT	ProCsysOrientMoveAxisOpt
PRO_E_CSYS_ORIENTSELAXIS2_FLIP	Flip second direction	-	
PRO_E_CSYS_ORIENTSELAXIS2_ROT_OPT	Second Axis Rotation Option	PRO_VALUE_TYPE_INT	ProCsysOrientSelAxisRotOpt
PRO_E_CSYS_ORIENTSELAXIS2_ROT	Second Axis Rotation	PRO_VALUE_TYPE_DOUBLE	Axisopt1 != AxisOpt2
PRO_E_CSYS_TYPE_MECH	Coordinate System Type (available in Creo Simulatemode only)	PRO_VALUE_TYPE_INT	ProCsysType
PRO_E_CSYS_FOLLOW_SRF_OPT	Follow Surface Option (available in Creo NC Sheetmetal mode only)	PRO_ELEM_TYPE_OPTION	ProCsysFollowSrfOpt
PRO_E_CSYS_NAME_DISPLAY_OPT	Name display option Specifies if the name of the coordinate system must be displayed in the graphics window. The valid values are defined in the enumerated data type ProCsysNameDisplayOpt:	PRO_VALUE_TYPE_INT	ProCsysNameDisplayOpt

Element Id	Element Name	Data Type	Valid Values
	• PRO_CSYS_NAME_DISPLAY_NO—This is the default value. Specifies that the name of the coordinate system must not be displayed in thw graphics window. • PRO_CSYS_NAME_DISPLAY_YES—Specifies that the name of the coordinate system must be displayed in the graphics window.		
PRO_E_CSYS_DISPLAY_ZOOM_DEP_OPT	Display zoom dependent option Specifies if the size of the coordinate system is dependent on the zoom of the model. The valid values are defined in the enumerated data type ProCsysDisplay ZoomDepOpt: • PRO_CSYS_DISPLAY_ZOOM_DEP_NO—This is the default value. Specifies that the coordinate system is independent of the zoom of the model. The coordinate system does not zoom when the model is zoomed. • PRO_CSYS_DISPLAY_ZOOM_DEP_YES—Specifies that the size of the coordinate system is dependent on the	PRO_VALUE_TYPE_INT	ProCsysDisplay ZoomDepOpt

Element Id	Element Name	Data Type	Valid Values
	zoom of the model. The coordinate system zooms when the model zooms.		
PRO_E_CSYS_AXIS_LENGTH	Axis length Specifies the default length for the coordinate system axes.	PRO_VALUE_TYPE_DOUBLE	

Note

To determine whether a coordinate system is a default coordinate system, query the number of PRO_E_CSYS_ORIGIN_CONSTRS and the number of PRO_E_CSYS_ORIENTMOVES. If both of the numbers are zero, then Csys is the default coordinate system.

The following elements are common for all the cases of the coordinate system feature creation:

Element ID	Value	Comments
PRO_E_FEATURE_TYPE	PRO_FEAT_CSYS	Mandatory
PRO_E_STD_FEATURE_NAME	Feature Name	Optional

Example 7: Creating a Datum Coordinate System

The sample code in the file UgGeneralCsysCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a General Csys at an offset to the specified Csys. The user is prompted to select a Csys.

Examples

Example 1: Using Three Planes or Two Edges and Axes

Use the following elements if the origin of the coordinate system is defined using three planes or using two edges and axes:

Element ID	Comments
PRO_E_CSYS_ORIGIN_CONSTRS	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS1_REF	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Mandatory if PRO_E_CSYS_ORIENTSELAXIS1_

Element ID	Comments
	REF is a Csys reference
PRO_E_CSYS_ORIENTSELAXIS1_FLIP	Optional
PRO_E_CSYS_ORIENTSELAXIS2_REF	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS2_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Mandatory if PRO_E_CSYS_ORIENTSELAXIS2_REF is a Csys reference
PRO_E_CSYS_ORIENTSELAXIS2_FLIP	Optional
Others	Not applicable

Example 2: Using Curve, Edges, or Plane and Axis

Use the following elements if the origin of the coordinate system is defined with a plane and an axis, curve, or edges:

Element ID	Comments
PRO_E_CSYS_ORIGIN_CONSTRS	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS1_REF	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Mandatory if PRO_E_CSYS_ORIENTSELAXIS1_REF is a Csys reference
PRO_E_CSYS_ORIENTSELAXIS1_FLIP	Optional
PRO_E_CSYS_ORIENTSELAXIS2_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS2_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Mandatory if PRO_E_CSYS_ORIENTSELAXIS2_REF is a Csys reference
PRO_E_CSYS_ORIENTSELAXIS2_FLIP	Optional
Others	Not applicable

Example 3: Using a Vertex or a Datum Point

Use the following elements if the origin of the coordinate system is defined using a vertex or a datum point:

Element ID	Comments
PRO_E_CSYS_ORIGIN_CONSTRS	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS1_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS1_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Mandatory if PRO_E_CSYS_ORIENTSELAXIS1_REF is a Csys reference
PRO_E_CSYS_ORIENTSELAXIS1_FLIP	Optional
PRO_E_CSYS_ORIENTSELAXIS2_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS2_REF_OPT	Optional, using default if not set

Element ID	Comments
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Mandatory if PRO_E_CSYS_ORIENTSELAXIS2_REF is a Csys reference
PRO_E_CSYS_ORIENTSELAXIS2_FLIP	Optional
Others	Not applicable

Example 4: Orienting by Selecting References

Use the following elements if PRO_E_CSYS_ORIENT_BY_METHOD is PRO_CSYS_ORIENT_BY_SEL_REFS:

Element ID	Comments
PRO_E_CSYS_ORIENTSELAXIS1_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS1_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS1_FLIP	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS2_REF	Mandatory
PRO_E_CSYS_ORIENTSELAXIS2_REF_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS2_OPT	Optional, using default if not set
PRO_E_CSYS_ORIENTSELAXIS2_FLIP	Optional, using default if not set
Others	Not applicable

Example 5: Orienting by Selecting Coordinate System Axes

Use the following elements if PRO_E_CSYS_ORIENT_BY_METHOD is PRO_CSYS_ORIENT_BY_SEL_CSYS_AXES:

Element ID	Comments
PRO_E_CSYS_NORMAL_TO_SCREEN	Optional, valid only if PRO_E_CSYS_ORIENT_BY_METHOD = PRO_CSYS_ORIENT_BY_SEL_CSYS_AXES. Otherwise, it is ignored.
Others	Not applicable

Example 6: Using a Coordinate System

Use the following elements if the origin of the coordinate system is determined using a Csys:

Element ID	Comments
PRO_E_CSYS_ORIGIN_CONSTRS	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR	Mandatory
PRO_E_CSYS_ORIGIN_CONSTR_REF	Mandatory
PRO_E_CSYS_OFFSET_TYPE	Optional, using default PRO_CSYS_OFFSET_CARTESIAN if not set
PRO_E_CSYS_ORIENTMOVES	Mandatory for non PRO_CSYS_OFFSET_CARTESIAN
PRO_E_CSYS_ORIENTMOVE	Mandatory for non PRO_CSYS_OFFSET_CARTESIAN

Element ID	Comments
PRO_E_CSYS_ORIENTMOVE_MOVE_TYPE	Mandatory for non PRO_CSYS_OFFSET_CARTESIAN
PRO_E_CSYS_ORIENTMOVE_MOVE_VAL	Mandatory for non PRO_CSYS_OFFSET_CARTESIAN
	For PRO_CSYS_OFFSET_CYLINDRICAL, the elements PRO_CSYS_ORIENTMOVE_MOVE_OPT_RAD, PRO_CSYS_ORIENTMOVE_MOVE_OPT_THETA, and PRO_CSYS_ORIENTMOVE_MOVE_OPT_ZI are required.
	For PRO_CSYS_OFFSET_SPHERICAL, the elements PRO_CSYS_ORIENTMOVE_MOVE_OPT_RAD, PRO_CSYS_ORIENTMOVE_MOVE_OPT_PHI, and PRO_CSYS_ORIENTMOVE_MOVE_OPT_THETA are required.
PRO_E_CSYS_ORIENT_BY_METHOD	Mandatory, using default PRO_CSYS_ORIENT_BY_SEL_REFS if not set

The function `ProDtmcsysTransformfileRead()` allocates required steps of the element tree to create CSYS from a transformation file.

The input file name to `ProDtmcsysTransformfileRead()` should have the name of a `.trf` file, with the extension. The name must be lowercase only. The file should contain a coordinate transform such as:

```
X1    X2    X3    Tx
Y1    Y2    Y3    Ty
Z1    Z2    Z3    Tz
```

where

- X1 Y1 Z1 is the X-axis direction,
- X2 Y2 Z2 is the Y-axis direction,
- X3 Y3 Z3 is not used (the right hand rule determines the Z direction), and
- Tx Ty Tz is the origin of the coordinate system.

35

Element Trees: Datum Curves

This chapter describes how to create, redefine, and access data for datum curve features using Creo Parametric TOOLKIT. The chapter Element Trees: Datum Features on page 804 provides necessary background for creating features; we recommend you read that material first.

Datum Curve Features

The element trees for datum curve features supported in Creo Parametric TOOLKIT are documented in the header file `ProDtmCrv.h`. Each datum feature type has a unique element tree containing the parameters and references necessary to create that type of feature.

Not all datum curve types are currently supported in Creo Parametric TOOLKIT. Some curve feature types are yet to be converted into element tree form. Other curve types have element trees with data that is not yet accessible through Creo Parametric TOOLKIT.

Common Elements

All datum curve features support the following common elements.

Element ID	Value
`PRO_E_FEATURE_TYPE`	`PRO_FEAT_CURVE`
`PRO_E_CURVE_TYPE`	As listed in ProCurveType. This element identifies the subtree to be used.
`PRO_E_STD_FEATURE_NAME`	Wstring (feature name)

Datum Curve Types

Creo Parametric TOOLKIT considers the following curve types for providing element tree access:

- Sketched Datum Curves on page 848
- Trim Datum Curves on page 849
- Intersect Datum Curves on page 849
- Wrap Datum Curves on page 850
- Offset Datum Curves on page 850
- Tangent Offset Datum Curves on page 851
- Datum Curves from Cross Section on page 851
- Datum Curves from Equation on page 852

Sketched Datum Curves

Creo Parametric TOOLKIT provides complete element tree access to the sketched datum curves. The sketched datum curves are sketched features, and therefore must be created using the techniques described in the chapter Element Trees: Sketched Features on page 1004.

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_SKETCHED
PRO_E_STD_SECTION	Section element tree
PRO_E_DTMCRV_DISPLAY_HATCH	Integer (PRO_B_TRUE, PRO_B_FALSE)
PRO_E_DTMCRV_HATCH_DENSITY	Double (if DISPLAY_HATCH = PRO_B_TRUE)

Trim Datum Curves

Creo Parametric TOOLKIT provides complete element tree access to trim datum curves (previously called Split datum curve).

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_SPLIT
PRO_E_STD_CRV_SPLIT_CURVE	The PRO_CURVE geometric item selected for splitting.
PRO_E_STD_CRV_DIVIDER	The geometric item used to divide the curve.
PRO_E_STD_CRV_SPLIT_SIDE	One of the ProSplit Sides enumerations

Intersect Datum Curves

Creo Parametric TOOLKIT provides complete element tree access to intersect datum curves. In the user interface, the intersect curve type results in one of the following curve types depending upon the references selected:

- A curve based on the intersection of two surfaces
- A curve based on the projections of two sections

The feature element tree for Intersect curve type contains two independent sets of elements to support both these feature types. The curve type determines which elements are required. As the two projections curve type contains two independent PRO_E_STD_SECTION elements, it must be created using the techniques described in the chapter Element Trees: Sketched Features on page 1004.

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_INTSRF
PRO_E_CRV_IP_REF_TYPE	PRO_CURVE_TYPE_INTSRF
PRO_E_CRV_IP_COMP_REF1	Compound
PRO_E_CRV_IP_REF_SEL1_TYPE	PRO_CURVE_TYPE_WHOLE for the whole surface selection; PRO_CURVE_TYPE_MULTIPLE_SEL for multiple independent surface selections.
PRO_E_CRV_IP_REF1	Based on the value of PRO_E_CRV_IP_SEL1_TYPE. If the value is whole, specifies a single selection of a datum plane, quilt, or solid geometry entity. If the value is multiple, specifies a multi-valued element containing any number of surface items.

Element ID	Value
PRO_E_CRV_IP_COMP_REF2	Compound
PRO_E_CRV_IP_REF_SEL2_TYPE	PRO_CURVE_TYPE_WHOLE for the whole surface selection; PRO_CURVE_TYPE_MULTIPLE_SEL for multiple independent surface selections.
PRO_E_CRV_IP_REF2	Based on the value of PRO_E_CRV_IP_SEL1_TYPE. If the value is whole, specifies a single selection of a datum plane, quilt, or solid geometry entity. If the value is multiple, specifies a multi valued element containing any number of surface items.
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_TWO_PROJ
PRO_E_CRV_IP_REF_TYPE	PRO_CURVE_TYPE_TWO_PROJ
PRO_E_CRV_IP_COMP_SEC1	Compound
PRO_E_STD_SECTION	Section element tree
PRO_E_CRV_IP_COMP_SEC2	Compound
PRO_E_STD_SECTION	Section element tree

Wrap Datum Curves

Creo Parametric TOOLKIT provides complete element tree access to wrap datum curves (also called Formed datum curves). Because the curve type contains a PRO_E_STD_SECTION element, you must create it using the techniques described in the chapter Element Trees: Sketched Features on page 1004.

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_WRAP
PRO_E_CRV_WRAP_SRF_TYPE	One of ProWrapSrfType
PRO_E_CRV_WRAP_SRF	Selection containing the wrap surface (surface, quilt or solid geometry)
PRO_E_STD_SECTION	Section
PRO_E_CRV_WRAP_FLIP	One of ProWrapFlip
PRO_E_CRV_WRAP_COORD_SYS	ID of the section coordinate system

Offset Datum Curves

Creo Parametric TOOLKIT provides partial element tree access to offset datum curves. In the user interface, the Offset curve type results in one of the following curve types depending upon the selected references:

- A curve offset normal to a surface
- A curve offset along a quilt

The feature element tree for Offset curve type contains elements that support both these feature types. The curve type determines which elements are required:

- PRO_CURVE_TYPE_OFFSET is offset normal to a surface
- PRO_CURVE_TYPE_OFFSET_IN_QUILT is offset along a quilt. Offset along a quilt is not supported in Creo Parametric TOOLKIT.

  Creation, redefinition or inspection of the curve type PRO_CURVE_TYPE_OFFSET_IN_QUILT is not supported. This is because the curve type contains elements that require data at run-time, which is not currently accessible to Creo Parametric TOOLKIT.

The following table lists all the elements that are used to create the curve type PRO_CURVE_TYPE_OFFSET.

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_OFFSET
PRO_E_CRV_OFFS_FEAT_TYPE	PRO_OFFSET_FROM_SURFACE
PRO_E_CRV_OFFS_SRF_REF	Selection of surface or quilt
PRO_E_CRV_OFFS_DIR_FLIP	One of ProOffsetDirFlip
PRO_E_DATUM_CURVE_OFFSET_VAL	The offset value or scale if a graph is used for offset
PRO_E_CRV_OFFS_CRV_REF	Selection of datum curve to be offset
PRO_E_CRV_OFFS_GRAPH_REF	Selection of graph used for offset calculation (optional)
PRO_E_CRV_OFFS_ST_END	One of ProOffsetStEnd

Tangent Offset Datum Curves

The curve type Tangent Offset is obsolete in Creo Parametric. As the existing models created in earlier releases may contain this curve type, Creo Parametric TOOLKIT provides read and redefine access only for these curves.

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_TANGENT_OFFSET
PRO_E_CRV_TANG_OFFSET_CURVE	Selection of curve to be offset
PRO_E_CRV_TANG_OFFSET_SURF	Selection of surface in which to create the offset
PRO_E_CRV_TANG_OFFSET_DIR	One of ProOffsetDirection
PRO_E_CRV_TANG_OFFSET_DIST	Offset value

Datum Curves from Cross Section

Creo Parametric TOOLKIT provides complete element tree access to datum curves created using existing planar cross sections in the model. The elements for this feature type are described below:

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_FROM_XSEC
PRO_E_CRV_FROM_XSEC_REF_XSEC	Mandatory element. Specifies the selection of a cross section to be used to create the datum curve.

Datum Curves from Equation

Creo Parametric TOOLKIT provides complete element tree access to datum curves created using an equation. Some of the elements for this feature type are described below:

Element ID	Value
PRO_E_CURVE_TYPE	PRO_CURVE_TYPE_FROM_EQUATION
PRO_E_CRV_FR_EQ_REF_CSYS	Selection of coordinate system used as a reference. The coordinate system represents the location of the 'zero point' of the equation.
PRO_E_CRV_FR_EQ_CSYS_TYPE	One of ProCrvFrEquatCsysTypes
PRO_E_CRV_FR_EQ_PARAM_MIN	Value of the lower limit for the domain of the curve
PRO_E_CRV_FR_EQ_PARAM_MAX	Value of the upper limit for the domain of the curve
PRO_E_CRV_ENTER_EQUATION	Parametric equation in terms the variables of the selected coordinate system type

Other Datum Curve Types

The following curve types contain run-time data in their element trees that is not currently accessible by Creo Parametric TOOLKIT. Currently, Creo Parametric TOOLKIT does not provide element tree access to the following curve types:

- Copy
- Project
- Boundary Offset

Some other curve types, including Thru Points, From File, and Use Xsec do not currently use element trees in Creo Parametric, and are therefore not accessible via Creo Parametric TOOLKIT.

36

Element Trees: Edit Menu Features

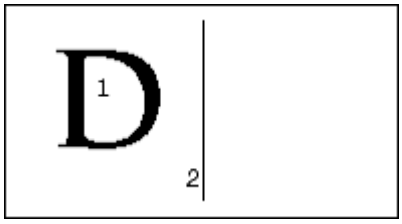
This chapter describes how to construct and access the element tree for some Edit Menu features in Creo Parametric TOOLKIT. It also shows how to redefine, create and access the properties of these features.

Mirror Feature

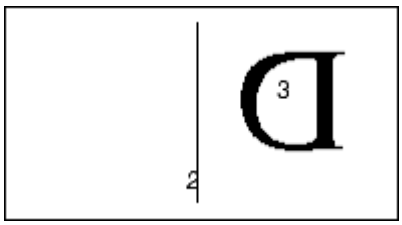
The Mirror feature creates a copy of the selected geometry by mirroring about a mirror plane. Creo Parametric TOOLKIT supports Mirror features where the initial selection of items was of geometry (curves and /or surfaces).Creo Parametric TOOLKIT does not support the element tree for Mirror features where the initial selection was of one or more features. Mirror features made from other features will have subfeatures listed under the Mirror entry in the Creo Parametric model tree, while Mirror features made from geometry will not include any sub-features.

Mirroring

Entity before Mirroring



Entity after Mirroring



where,

1—Specifies original entity such as plane, surfaces, axes or parts.

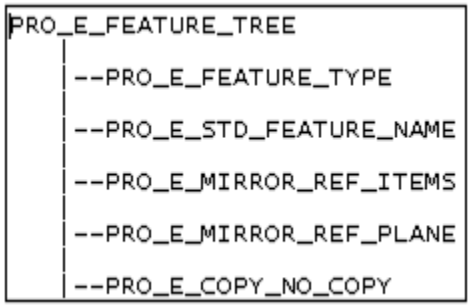
2—Specifies mirror line (mirror plane).

3—Specifies the copy of the original entity.

The Feature Element Tree for Mirror feature in Creo Parametric

The element tree for a Mirror feature is documented in the header file `ProMirror.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Mirror Feature



The Mirror element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—Must be `PRO_FEAT_SRF_MDL`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the feature name.
- `PRO_E_MIRROR_REF_ITEMS`—A multivalued element that includes the mirror item reference. It must be references of the following types: `PRO_CURVE`, `PRO_COMP_CRV`, `PRO_AXIS`, `PRO_QUILT`, `PRO_PART`.
- `PRO_E_MIRROR_REF_PLANE`—Specifies the mirror plane and is a mandatory element and must be `PRO_DATUM_PLANE` or `PRO_SURFACE` (only planar surfaces).
- `PRO_E_COPY_NO_COPY`—Specifies the option to create a copy or not and is a mandatory element. It can have either of the following values:
  - `PRO_MIRROR_KEEP_ORIGINAL`
  - `PRO_MIRROR_HIDE_ORIGINAL`

Creating a Mirror Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Mirror Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Mirror Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Mirror Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Mirror Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Mirror Feature and to retrieve the element tree description of a Mirror Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Move Feature

The Move feature can be used to create and move a copy of an existing surface or curve rather than moving the original. Using the Move feature saves time because it enables you to create simple patterns with surfaces and curves.

The Move feature is available for both part and assembly modes. You can apply this feature in different modes:

- Translate mode—Translate surfaces, datum curves, and axes in a direction perpendicular to a reference plane. You can also translate along a linear edge, axis, perpendicular to plane, two points, or coordinate system.
- Rotate mode—Rotate surfaces, datum curves, and axes about an existing axis, linear edge, perpendicular to plane, two points, or coordinate system.

To move a surface or curve relative to its original position, define a move reference. In the Translate mode, the move reference is typically a plane or edge that is perpendicular to the direction in which you want to translate the moved feature. In Rotate mode, the move reference is typically an axis or edge about which you want to rotate the moved feature. In this mode, the moved object moves about the direction reference.

Following types of move references exists in Translate mode:

- Linear curve
- Linear edge
- Axis
- Axis of a coordinate system

- Plane
- Two points

> **Note**
>
> The direction reference is always perpendicular to the direction in which you want to move.

Following types of move references exists in Rotate mode:

- Linear curve
- Edge
- Axis
- Axis of a coordinate system
- Two points

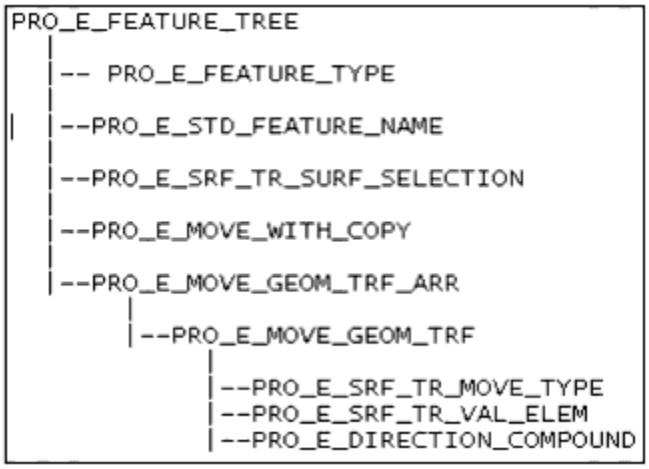
> **Note**
>
> You can flip the direction of the move in the Rotate mode.

Creo Parametric TOOLKIT supports Move features where the initial selection of items was of geometry (curves and /or surfaces). Creo Parametric TOOLKIT does not support the element tree for Move features where the initial selection was of one or more features. Move features made from other features will have subfeatures listed under the Move entry in the Creo Parametric model tree, while Move features made from geometry will not include any sub-features.

The Feature Element Tree for Move feature in Creo Parametric

The element tree for a Move feature is documented in the header file `ProMove.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Move Feature



The move element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—Must be `PRO_FEAT_SRF_MDL`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the feature name.
- `PRO_E_SRF_TR_SURF_SELECTION`—Specifies whether to move or copy the selected geometry. Curves, composite curves, axes and quilts are eligible to for this element.
- `PRO_E_MOVE_NO_COPY`—Specifies an option to control copy of the original geometry.
- Provides the ability to transform geometry with or without a copy.
- `PRO_E_MOVE_WITH_COPY`—Specifies whether to hide or display the original geometry after copy.
- `PRO_E_MOVE_GEOM_TRF_ARR`—Contains an array of movements applied to the selected entities. This can be a combination of translational and rotational transformations.
- `PRO_E_DIRECTION_COMPOUND`—Specifies the direction reference for the translation or rotational movement. This compound element is a standard Creo Parametric element subtree and is described in `ProDirection.h`.

Creating a Move Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Move Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Move Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Move Feature based on the changes made in the element tree. For more information about`ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Move Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Move Feature and to retrieve the element tree description of a Move Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Fill Feature

Creo Parametric TOOLKIT enables you to create and redefine flat surface features called fill features. A fill feature is simply a flat surface feature that is defined by its boundaries and is used to thicken surfaces.

Note

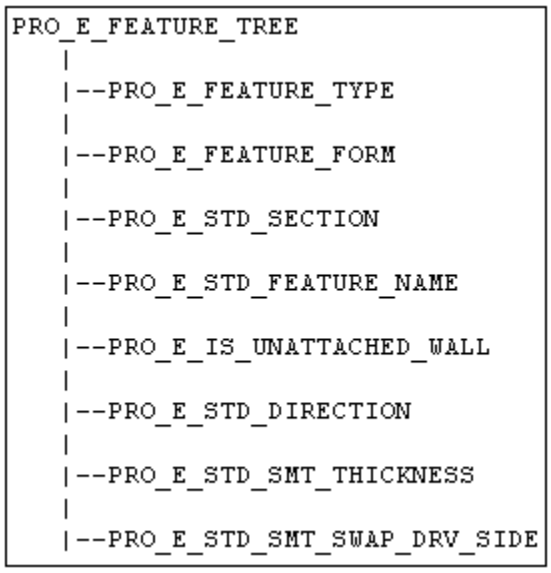
All fill features must consist of a flat, closed-loop sketched feature.

The Feature Element Tree for Fill feature in Creo Parametric

The element tree for a Fill feature is documented in the header file `ProFlatSrf.h`, and has a simple structure.

The following figure demonstrates the feature element tree structure:

Feature Element Tree for Fill Feature



The fill element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—It should be `PRO_FEAT_DATUM_SURF`.
- `PRO_E_FEATURE_FORM`—Specifies feature form and should be of `PRO_FLAT` type only.
- `PRO_E_STD_SECTION`—Specifies a sketched section. Refer to the section Creating Features Containing Sections on page 1006 for details on how to create features that contain sketched sections.

The element tree for the Fill feature also supports access to Flat Sheetmetal Wall features. Refer to the chapter Production Applications: Sheetmetal on page 1310 for element details.

Creating a Fill Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Fill Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Fill Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Fill Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Fill Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Fill Feature and to retrieve the element tree description of a Fill Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Intersect Feature

Refer to the section Intersect Datum Curves on page 849 in the chapter Element Trees: Datum Curves on page 847 for details about this feature.

Merge Feature

The Merge Feature is created by merging two or more selected quilts. In case of more than two quilts, every input quilt should have at least one of its edges adjacent to the edge of any other input quilt, and the surfaces must not overlap.

You can merge a number of input quilts by joining two adjacent quilts one after another, that is, by aligning the edges of one quilt to the edges of the other. The first quilt selected for the merge operation becomes the primary reference quilt. The second adjacent quilt is joined to the primary quilt, forming the main body or a newly formed primary quilt. The third quilt is, then, joined to the main body. This process continues until all the input quilts are joined together.

Note

- For a successful merge, the selected quilts must be ordered based upon their adjacency.
- The merge operation fails in case of intersecting quilts, and quilts that are not adjacent to any other input quilt. In either case, remove the problematic quilt to complete the merge.

You can also select the input quilts for the merge operation using region selection. In case of region selection, only box selection is available and the quilts to be selected must be completely inside the region. The selected quilts are ordered based on the feature number of the quilt's parent feature.

You can merge only two quilts by intersecting. You can specify which portion of the quilt to include in the merge feature by selecting the sides for each of the quilts.

A merged quilt consists of two or more original quilts that provide the geometry, and a merge feature that contains the information for the surface joining. The input quilts are retained, even if you delete the Merge feature.

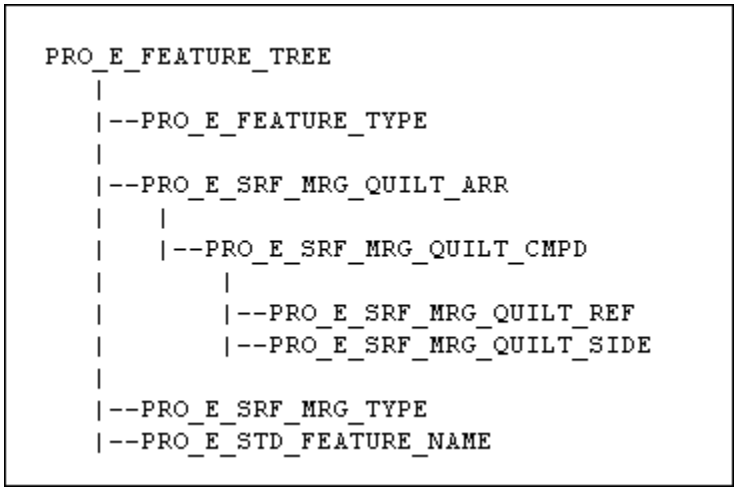
Note

In Assembly mode, you can merge only assembly-level quilts. If you want to create component-level merge features, you must first activate the component, and then merge the quilts in the component. Surface merge is available only for surfaces that belong to the same component.

Feature Element Tree for Merge feature in Creo Parametric

The element tree for the Merge feature is documented in the header file `ProMerge.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Merge Feature



The following list details special information about the elements in this tree:

- PRO_E_FEATURE_TYPE—Specifies the feature type; must be PRO_FEAT_DATUM_QUILT.
- PRO_E_SRF_MRG_QUILT_ARR—Specifies an array of the following compound element. The array must have atleast two elements to create the Merge feature.
  - PRO_E_SRF_MRG_QUILT_CMPD—Specifies the set of references and sides to be used for the merge operation. The set consists of the following two elements:
    - PRO_E_SRF_MRG_QUILT_REF—Specifies the quilt of the type PRO_QUILT selected for the merge operation. This is a mandatory element.
    - PRO_E_SRF_MRG_QUILT_SIDE—Specifies the side of the selected quilt that should be included in the merge feature in case of intersecting quilts. This element is ignored when the quilt array PRO_E_SRF_MRG_QUILT_ARR contains more than two elements.
- PRO_E_SRF_MRG_TYPE—Specifies the merge type and can have following merge type options:
  - PRO_SRF_MRG_JOIN—For joining quilts.
  - PRO_SRF_MRG_INTSCT—For merging two quilts at the intersection. This is the default option.

  The merge type must be PRO_SRF_MRG_JOIN when the quilt array PRO_E_SRF_MRG_QUILT_ARR contains more than two elements.
- PRO_E_STD_FEATURE_NAME—Specifies the merge feature name.

Creating a Merge Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Merge Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Merge Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Merge Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Merge Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Merge Feature and to retrieve the element tree description of a Merge Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Pattern Feature

Refer to the chapter Element Trees: Patterns on page 963 for details about this feature.

Wrap Feature

Refer to the section Wrap Datum Curves on page 850 in the chapter Element Trees: Datum Curves on page 847 for details about this feature.

Trim Feature

Trim feature is applied to remove portions of an existing surface feature. A trim feature is similar to a solid cut. Following are the existing types of feature creation for the trimming of a surface or a quilt.

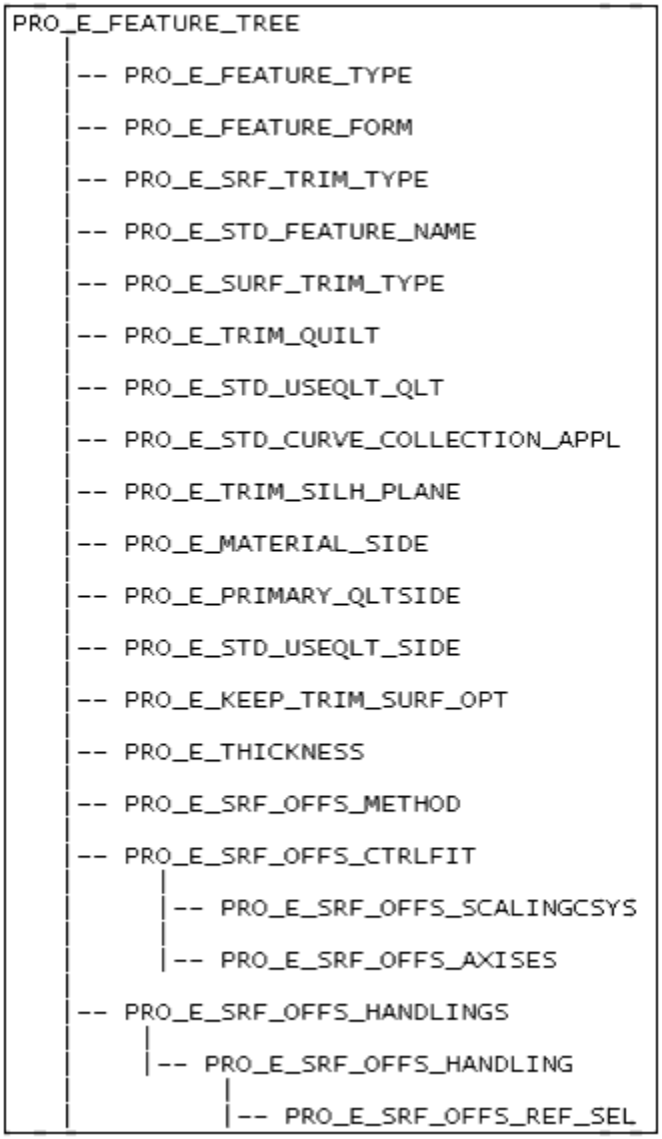
- Use Quilts—Cuts a piece from a surface using an intersecting quilt. Creo Parametric consumes the quilt that is used to trim a surface and allows you to keep either or both sides of the trimmed surface. You can trim quilts in the following ways:
  - By adding a cut or slot as you do to remove material from solid features
  - By trimming the quilt at its intersection with another quilt or to its own silhouette edge as it appears in a certain view
  - By filleting corners of the quilt
  - By trimming along a datum curve lying on the quilt
- Use Curves—Trims a surface using selected curves and edges.

  The rules for defining a surface trim using a datum curve are as follows:
  - You can use a continuous chain of datum curves, inner surface edges, or solid model edges to trim a quilt.
  - Datum curves used for trimming must lie on the quilt to be trimmed and should not extend beyond the boundaries of this quilt.
  - If the curve does not extend to the boundaries of the quilt, the system calculates the shortest distance to the quilt boundary and continues the trim in this direction.
- Silhouette—Trims a surface at its silhouette edge from a specified direction.

The Feature Element Tree for Trim feature in Creo Parametric

The element tree for a Trim feature is documented in the header file `ProTrim.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Trim Feature



The trim element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—It should be `PRO_FEAT_CUT`.
- `PRO_E_FEATURE_FORM`—Specifies feature form and is a mandatory element applicable only for use quilt and thin types. It has `PRO_USE_SURFS` and `PRO_NOTYPE` as their valid values.
- `PRO_E_SRF_TRIM_TYPE`—Specifies trim type identity and is a non-redefinable mandatory element. The trim type determines the required

structure for the remaining elements in the element tree. Each type requires different mandatory elements. The different trim types are as follows:

  - PRO_SURF_TRIM_USE_CRV—use curves to trim the quilt.
  - PRO_SURF_TRIM_USE_QLT—use another quilt to trim the given quilt.
  - PRO_SURF_TRIM_THIN
  - PRO_SURF_TRIM_SILH—trim quilt using silhouette edges.
- PRO_E_TRIM_QUILT—Specifies trimmed quilt and should be of type PRO_SURFACE or PRO_QUILT only.
- PRO_E_STD_USEQLT_QLT—Specifies trimming quilts and trimming planes. It should have PRO_SURFACE, PRO_QUILT and PRO_DATUM_PLANE values only. It is applicable for use quilt and thin types only.
- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the trimming curves and is applicable for use curve only.
- PRO_E_TRIM_SILH_PLANE—Specifies silhouette plane and can be of type PRO_SURFACE or PRO_DATUM_PLANE only.
- PRO_E_MATERIAL_SIDE—Specifies material side options and has the following values:
  - PRO_TRIM_MATL_SIDE_ONE
  - PRO_TRIM_MATL_SIDE_TWO
  - PRO_TRIM_MATL_BOTH_SIDES
- PRO_E_PRIMARY_QLTSIDE—Specifies the primary quilt side options and they are listed as follows:
  - PRO_TRIM_PRIM_QLT_SIDE_ONE
  - PRO_TRIM_PRIM_QLT_SIDE_TWO
- PRO_E_STD_USEQLT_SIDE—Specifies thickness direction options for thin trims and can only be as follows:
  - PRO_TRIM_STD_QUILT_SIDE_ONE
  - PRO_TRIM_STD_QUILT_SIDE_TWO
  - PRO_TRIM_STD_QUILT_BOTH_SIDES
- PRO_E_KEEP_TRIM_SURF_OPT—Specifies to keep trimming surface option and is valid only when the trimming quilt is a surface or quilt. It has the following values:
  - PRO_KEEP_TRIM_SURF_OPT_NO
  - PRO_KEEP_TRIM_SURF_OPT_YES
- PRO_E_THICKNESS—Specifies thickness for thin trims.

- PRO_E_SRF_OFFS_METHOD—Specifies the types of offset for thin trims and are as follows:
  - PRO_TRIM_SRF_OFFS_METH_NORMTOSURF—The surface or quilt is thickened in a direction normal to the surface.
  - PRO_TRIM_SRF_OFFS_METH_AUTOSCALE—The surface or quilt is thickened by automatically determining scaling coordinate system and fit along all three axes.
  - PRO_TRIM_SRF_OFFS_METH_MANUALSCALE—The surface or quilt is thickened by specifying the scaling coordinate system and control fitting motion.
- PRO_E_SRF_OFFS_CTRLFIT—Specifies the control fit and is applicable for thin trims only. It offsets the surface by creating a best-fit offset that is scaled with respect to a selected coordinate system. It allows you to define the axis for translation. It consists of the following elements:
  - PRO_E_SRF_OFFS_SCALINGCSYS specifies the control fit coordinate system and should have a value of PRO_CSYS.
  - PRO_E_SRF_OFFS_AXISES specifies the control fit axes having the following values:
    - PRO_TRIM_OFFS_TRF_NONE
    - PRO_TRIM_OFFS_TRF_X
    - PRO_TRIM_OFFS_TRF_Y
    - PRO_TRIM_OFFS_TRF_Z
    - PRO_TRIM_OFFS_TRF_XY
    - PRO_TRIM_OFFS_TRF_YZ
    - PRO_TRIM_OFFS_TRF_ZX
    - PRO_TRIM_OFFS_TRF_ALL
- PRO_E_SRF_OFFS_HANDLINGS—Specifies an array of PRO_E_SRF_OFFS_HANDLING and is applicable for thin trims only.
- PRO_E_SRF_OFFS_HANDLING—Specifies special handling item and consists of PRO_E_SRF_OFFS_REF_SEL which are special handling faces which should be from the trimming quilt.

A list of elements required for different types of trims:

Trim type	Element Id
Quilt Type	PRO_E_FEATURE_TYPE
	PRO_E_FEATURE_FORM
	PRO_E_SRF_TRIM_TYPE
	PRO_E_STD_FEATURE_NAME

Trim type	Element Id
	PRO_E_SURF_TRIM_TYPE
	PRO_E_TRIM_QUILT
	PRO_E_STD_USEQLT_QLT
	PRO_E_MATERIAL_SIDE
	PRO_E_PRIMARY_QLTSIDE
	PRO_E_KEEP_TRIM_SURF_OPT
Use Curve	PRO_E_FEATURE_TYPE
	PRO_E_SRF_TRIM_TYPE
	PRO_E_STD_FEATURE_NAME
	PRO_E_SURF_TRIM_TYPE
	PRO_E_TRIM_QUILT
	PRO_E_STD_CURVE_COLLECTION_APPL
	PRO_E_MATERIAL_SIDE
	PRO_E_PRIMARY_QLTSIDE
Thin	PRO_E_FEATURE_TYPE
	PRO_E_FEATURE_FORM
	PRO_E_SRF_TRIM_TYPE
	PRO_E_STD_FEATURE_NAME
	PRO_E_SURF_TRIM_TYPE
	PRO_E_TRIM_QUILT
	PRO_E_STD_USEQLT_QLT
	PRO_E_STD_USEQLT_SIDE
	PRO_E_KEEP_TRIM_SURF_OPT
	PRO_E_THICKNESS
	PRO_E_SRF_OFFS_METHOD
	PRO_E_SRF_OFFS_CTRLFIT
	PRO_E_SRF_OFFS_HANDLINGS
Silhouette	PRO_E_FEATURE_TYPE
	PRO_E_SRF_TRIM_TYPE
	PRO_E_STD_FEATURE_NAME
	PRO_E_SURF_TRIM_TYPE
	PRO_E_TRIM_QUILT
	PRO_E_TRIM_SILH_PLANE
	PRO_E_MATERIAL_SIDE
	PRO_E_PRIMARY_QLTSIDE
	PRO_E_KEEP_TRIM_SURF_OPT

Creating a Trim Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Trim Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Trim Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Trim Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Trim Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Trim Feature and to retrieve the element tree description of a Trim Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Offset Feature

Refer to the section Offset Datum Curves on page 850 in the chapter Element Trees: Datum Curves on page 847 for details about this feature.

Thicken Feature

The Thicken feature is available for both part and for assembly modes and can be defined with respect to coordinate systems, axes, and surfaces. You can apply this feature to check the thickness of a part in the model. This feature also forms the basis for model analysis and thickness check. The result of the thickness analysis is as follows:

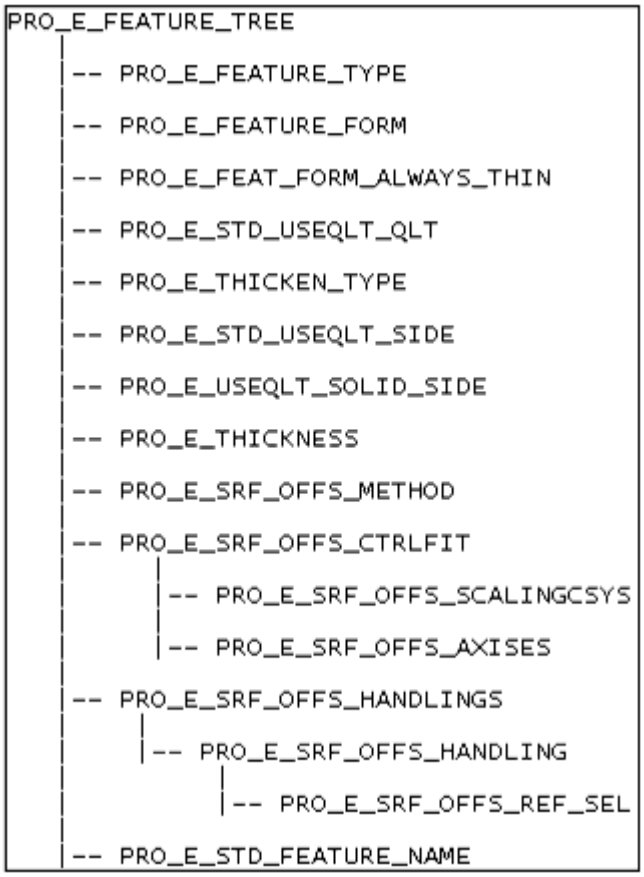
- The thickness is between the design range.
- The thickness exceeds the maximum design value.
- The thickness is below the minimum design value.

Thicken features use predetermined surface features or quilt geometry to add or remove thin material sections in the design. The surface features or quilt geometry provide greater flexibility within the design and enable you to transform the geometry to better meet the design needs.

The Feature Element Tree for Thicken feature in Creo Parametric

The element tree for a Thicken feature is documented in the header file `ProThicken.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Thicken Feature



The Thicken element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—`PRO_FEAT_PROTRUSION` or `PRO_FEAT_CUT`.
- `PRO_E_FEATURE_FORM`—Must be `PRO_E_SURFS`.

- PRO_E_FEAT_FORM_ALWAYS_THIN—Must be PRO_THIN.
- PRO_SRF_OFFS_METHOD—Offset method that can be enumerated as follows:
  - PRO_OFFS_METH_NORMTOSURF— Specifies the offset of the thickened surface normal to the original surface.
  - PRO_OFFS_METH_AUTOSCALE— Specifies autoscale and translates the thickened surface with respect to an automatically determined coordinate system.
  - PRO_OFFS_METH_MANUALSCALE— Specifies manual scale.

Note

If you change the offset method for a particular feature, all children of this feature fail.

Creating a Thicken Feature

Function Introduced

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a Thicken Feature based on element tree input. For more information about ProFeatureCreate(), refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Thicken Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function ProFeatureRedefine() to redefine a Thicken Feature based on the changes made in the element tree. For more information about ProFeatureRedefine(), refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Thicken Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Thicken Feature and to retrieve the element tree description of a Thicken Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter . Element Trees: Principles of Feature Creation on page 764

Solidify Feature

Solidify features are used to create complex geometry that would be difficult to create using regular solid features.

Solidify features use predetermined surface features or quilt geometry and convert them into solid geometry. You can use Solidify features to add, remove, or replace solid material in the designs. The quilt geometry provides greater flexibility within the design, and the Solidify feature enables the designer to transform the geometry to meet design needs.

The attributes of the Solidify feature are:

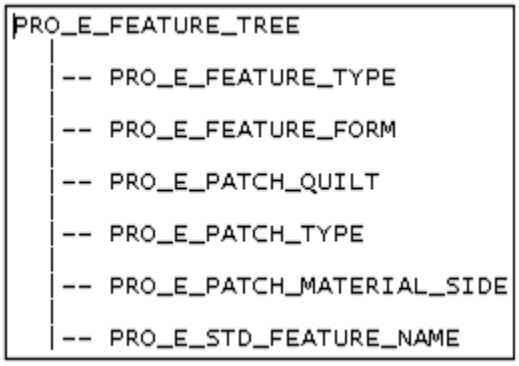
- Patch—Replaces a portion of the surface with quilt. The quilt boundary must lie on the surface.
- Add Material—Fills a quilt with solid material.
- Remove Material—Removes material from inside or outside a quilt.

> **Note**
>
> add material and remove material attributes are always available. Patch is available only if the selected quilt boundaries lie on solid geometry.

The Feature Element Tree for Solidify Feature in Creo Parametric

The element tree for a Solidify feature is documented in the header file `ProSolidify.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Solidify Feature



The solidify element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_STD_FEATURE_TYPE`—Must be one of the following types—
  - Protrusion—`PRO_FEAT_PROTRUSION`
  - Cut—`PRO_FEAT_CUT`
  - Patch—`PRO_FEAT_PATCH`
- `PRO_E_FEATURE_FORM`— Must be of the following types:
  - `PRO_USE_SURFS`— Represents protrusion (`PRO_FEAT_PROTRUSION`) and cut (`PRO_FEAT_CUT`).
  - `PRO_NOTYPE`— Represents patch (`PRO_FEAT_PATCH`).
- `PRO_E_PATCH_QUILT`— Specifies the reference quilt and is a mandatory element. A valid quilt or surface satisfies any of the following contexts:

`Quilt is open`

- CASE 1
  - All boundaries lie on solid surfaces.
  - Solid geometry does not intersect quilt.
  - Solid geometry and quilt form closed empty volume.
- CASE 2
  - All boundaries lie on solid surfaces.
  - Solid geometry intersects quilt.
- CASE 3
  - All one sided edges are inside solid geometry.
- CASE 4
  - All one sided edges are outside solid geometry.

    - Solid geometry intersects quilt.
- CASE 5
    - Solid geometry and quilt form closed empty volume.
    - At least, one sided edge intersects geometry.
- CASE 6
    - Quilt intersects geometry.
    - None of the one sided edges intersect geometry.
- CASE 7
    - Solid geometry does not intersect quilt.

`Quilt is closed`

- CASE 1
    - Solid geometry does not intersect quilt.
- CASE 2
    - Quilt completely buried inside Material.
- CASE 3
    - Solid geometry partly intersects quilt.

Creating a Solidify Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Solidify Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Solidify Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Solidify Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Solidify Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Solidify Feature and to retrieve the element tree description of a Solidify Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Remove Feature

Use the Remove feature to remove geometry without disturbing the feature history. It allows you to modify existing imported geometry or delete some geometry from a part (not necessarily formed by a single feature) without having to reroute and refine a number of features.

You can select one of the following items for removal:

- Surfaces, surface sets, or intent surfaces
- One closed loop chain of one-sided edges from a single quilt

Geometry removal results in neighboring surfaces being extended or trimmed to converge and close the void. The extended geometry is attached as a solid or surface to the selected surfaces. In case of a one-sided edge chain selected as the reference, the extended geometry can be attached to the selected quilt, or created as a new quilt.

You can exclude contours in multi-contour surfaces from being removed. For periodic features separated by artificial edges (for example, extruded cylinders or revolved features), the feature internally references all the surfaces, even if one of them is selected.

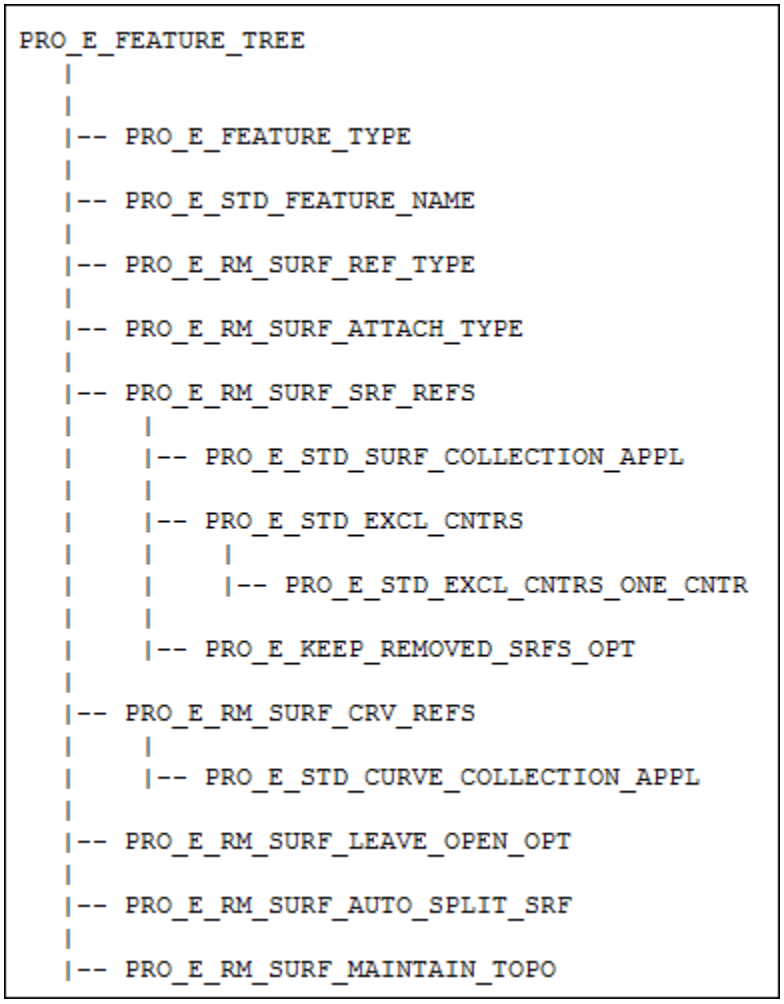
Note

- All surfaces that are extended or trimmed must be adjacent to the boundary defined by references.
- If an adjacent surface does not need to be extended, it will not be copied.
- Surfaces that are to be extended must be extendable.
- Extended surfaces must converge to form a defined volume.
- No new patches are created when the surfaces are extended.
- If the modified geometry cannot be attached as a solid, it can be manually attached as a quilt.
- Feature operations such as Suppress, Delete, Redefine, Reroute, Copy/Paste, and Pattern (limited to reference pattern if patterning by itself) are supported.
- Transformation of this feature by itself is not applicable and allowed.

Feature Element Tree for the Remove Feature

The element tree for the Remove feature is documented in the header file `ProRemoveSurf.h`. The following figure demonstrates the element tree structure:

Feature Element Tree for Remove Feature



The Remove element tree contains standard element types. The following list details special information about the elements in this tree:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should always have the value `PRO_FEAT_RM_SURF`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the remove feature.
- `PRO_E_RM_SURF_REF_TYPE`—Specifies the reference type. This element decides the type of references that you can select for removal. It can have the following values:
  - `PRO_RM_SURF_SRF_REF`—Specifies that surface sets can be selected as the references.
  - `PRO_RM_SURF_CRV_REF`—Specifies that a chain of one-sided edges can be selected as the reference.

- PRO_E_RM_SURF_ATTACH_TYPE—Specifies the attachment type. It can have the following values:
  - FM_RM_SURF_ATTACH_SAME—Specifies that the extended geometry will be attached to the selected surfaces if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_SRF_REF, or to the selected quilt if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_CRV_REF.
  - FM_RM_SURF_ATTACH_SEP—Specifies that the extended geometry will be created as a separate quilt if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_CRV_REF.
- PRO_E_RM_SURF_SRF_REFS—Specifies the set of surface references. This element is required if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_SRF_REF. It consists of the following elements:
  - PRO_E_STD_SURF_COLLECTION_APPL—Specifies the surfaces selected for removal. You can select multiple surfaces. This element is required if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_SRF_REF.
  - PRO_E_STD_EXCL_CNTRS—Specifies an array of excluded contours of the type PRO_E_STD_EXCL_CNTRS_ONE_CNTR.
  - PRO_E_KEEP_REMOVED_SRFS_OPT—Specifies the ProBoolean option that allows you to retain the removed surfaces as a separate quilt.
- PRO_E_RM_SURF_CRV_REFS—Specifies the set of curve references. This element is required if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_CRV_REF. It consists of the following element:
  - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the single closed loop chain of one-sided edges. This element is required if PRO_E_RM_SURF_REF_TYPE is equal to PRO_RM_SURF_CRV_REF.
- PRO_E_RM_SURF_LEAVE_OPEN_OPT—Specifies if the hole that is created after surfaces are removed must be kept open. The solids are converted to quilts and closed quilts are converted to open quilts.
- PRO_E_RM_SURF_AUTO_SPLIT_SRF—Specifies if the extending surface for the selected shape surface must be automatically split when geometry is removed. The splitting surface is automatically selected.
- PRO_E_RM_SURF_MAINTAIN_TOPO—Specifies if the regeneration must fail when the model changes and the same solution type cannot be reconstructed. It is used to indicate if the topology in the model must be maintained.

Note

The Remove feature is available in the parametric modeling environment as well as in the flexible modeling environment. In the parametric modeling environment, a collection of surfaces or curves are used as the references. In the flexible modeling environment, a region or a collection of surfaces that include regions is used as the reference. In the flexible modeling environment with a surface collection that includes regions, the Remove feature replaces each region by the surface it belongs to. The contours that define other regions of the same surface are automatically saved as the `PRO_E_STD_EXCL_CNTRS_ONE_CNTR` element.

Element Details of PRO_E_STD_EXCL_CNTRS_ONE_CNTR

Each `PRO_E_STD_EXCL_CNTRS_ONE_CNTR` specifies one excluded contour. It consists of the following elements:

PRO_E_STD_EXCL_CNTRS_ONE_CNTR

```
PRO_E_STD_EXCL_CNTRS_ONE_CNTR
 |
 |-- PRO_E_STD_EXCL_CNTR_SRF_REF
 |
 |-- PRO_E_STD_EXCL_CNTR_EDGE_REF
```

- `PRO_E_STD_EXCL_CNTR_SRF_REF`—Specifies the surface reference for the excluded contour.
- `PRO_E_STD_EXCL_CNTR_EDGE_REF`—Specifies the edge reference for the excluded contour. This element can be any edge of the contour.

Creating the Remove Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create the Remove feature based on the element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 in the chapter Element Trees: Principles of Feature Creation on page 764.

Redefining the Remove Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine the Remove feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing the Remove Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of the Remove feature, and to retrieve the element tree description of the Remove feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

Example 1: Creating a Remove Surface Feature

The sample code in `UgRemoveSurfCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a remove Surface feature.

Attach Feature

The Attach feature allows you to attach open quilts to solid or quilt geometry, if the open quilt does not intersect the solid or quilt geometry. You can select an open quilt and attach it to another quilt or solid geometry within the same model. You can also select two open quilts within the same model which do not intersect and attach them.

This feature is useful in case of UDF placement when the geometry of the UDF does not intersect the part.

Feature Element Tree for Attach Feature

The element tree for a Attach feature is documented in the header file `ProFlexAttach.h` and has a simple structure. The following figure demonstrates the structure of the feature element tree.

Attach Feature Element Tree

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_REF_ATTACH_PRIM_QLT
   |
   |--PRO_E_REF_ATTACH_MERG_QLT
   |
   |--PRO_E_OPT_ATTACH_OPER
   |
   |--PRO_E_OPT_ATTACH_RMV_MAT
   |
   |--PRO_E_OPT_ATTACH_PQ_DIR
   |
   |--PRO_E_OPT_ATTACH_MGQ_DIR
   |
   |--PRO_E_OPT_ATTACH_PIO
   |
   |--PRO_E_OPT_ATTACH_RNDCH
   |
   |--PRO_E_FLEX_OPTS_CMPND
```

The elements in this tree are described below:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_FLXATTACH`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature.
- `PRO_E_ATTACH_QLT_REFS`—Specifies the open quilts you select for attachment.
- `PRO_E_ATTACH_OPT_TYPE`—Specifies the attachment type. It is given by the enumerated type `ProFlexAttachOptType` and is of the following types:
  - `PRO_FLXATT_SOLID_OPT`—Specifies the solid to which the open quilts are attached.
  - `PRO_FLXATT_SURF_OPT`—Specifies the quilt surface to which the open quilts are attached.

Creating the Attach Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create the Attach feature based on the element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 in the chapter Element Trees: Principles of Feature Creation on page 764.

Redefining the Attach Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine the Attach feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing the Attach Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to to retrieve the element tree description of the Attach feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

37

Element Trees: Replace

This chapter describes the basic principles of creating a tweak surface replacement feature. The chapter Element Trees: Principles of Feature Creation on page 764 is a necessary background for this topic. Read that chapter before this one.

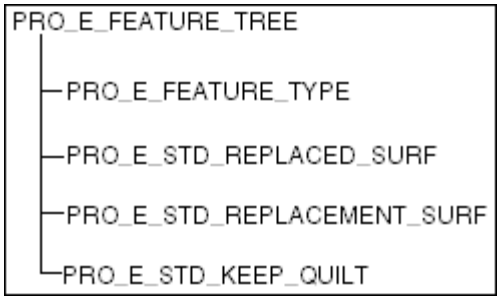
Introduction

The surface replacement functionality enables you to replace the specified surface on a model with a datum plane or quilt. This is similar to the **Replace** option from **TWEAK** menu in Creo Parametric. See the corresponding section in the *Part Modeling User's Guide* for a detailed description of the restrictions and requirements of the feature.

The Feature Element Tree

The element tree for a surface replacement feature is documented in the header file `ProReplace.h`, and is shown in the following figure.

Element Tree for Surface Replacement



The following table describes the elements in the element tree for the surface replacement feature.

Element ID	Data Type	Description
`PRO_E_FEATURE_TYPE`	`PRO_VALUE_TYPE_INT`	`PRO_FEAT_REPLACE_SURF`
`PRO_E_STD_REPLACED_SURF`	`PRO_VALUE_TYPE_ SELECTION`	The surface to be removed
`PRO_E_STD_REPLACEMENT_ SURF`	`PRO_VALUE_TYPE_ SELECTION`	The replacement surface
`PRO_E_STD_KEEP_QUILT`	`PRO_VALUE_TYPE_INT`	Specifies whether to keep the quilt

To keep the replacement surface (datum plane or quilt), add the element `PRO_E_ STD_KEEP_QUILT` and set its value to 1. If you omit the element, or its value is 0, the replacement surface will be consumed by the replacement feature.

After you have defined the element tree, call the function `ProFeatureCreate()` to create the tweak surface replacement feature.

38

Element Trees: Draft Features

This chapter introduces and shows how to create, redefine and access draft features.

Draft Feature

The Draft feature adds a draft angle between -89.9° and +89.9° to individual surfaces or to a series of surfaces.

You can draft either solid surfaces or quilt surfaces, but not a combination of both. The first selected surface determines the type of additional surfaces (solid or quilts) that can be selected as draft surfaces for this feature.

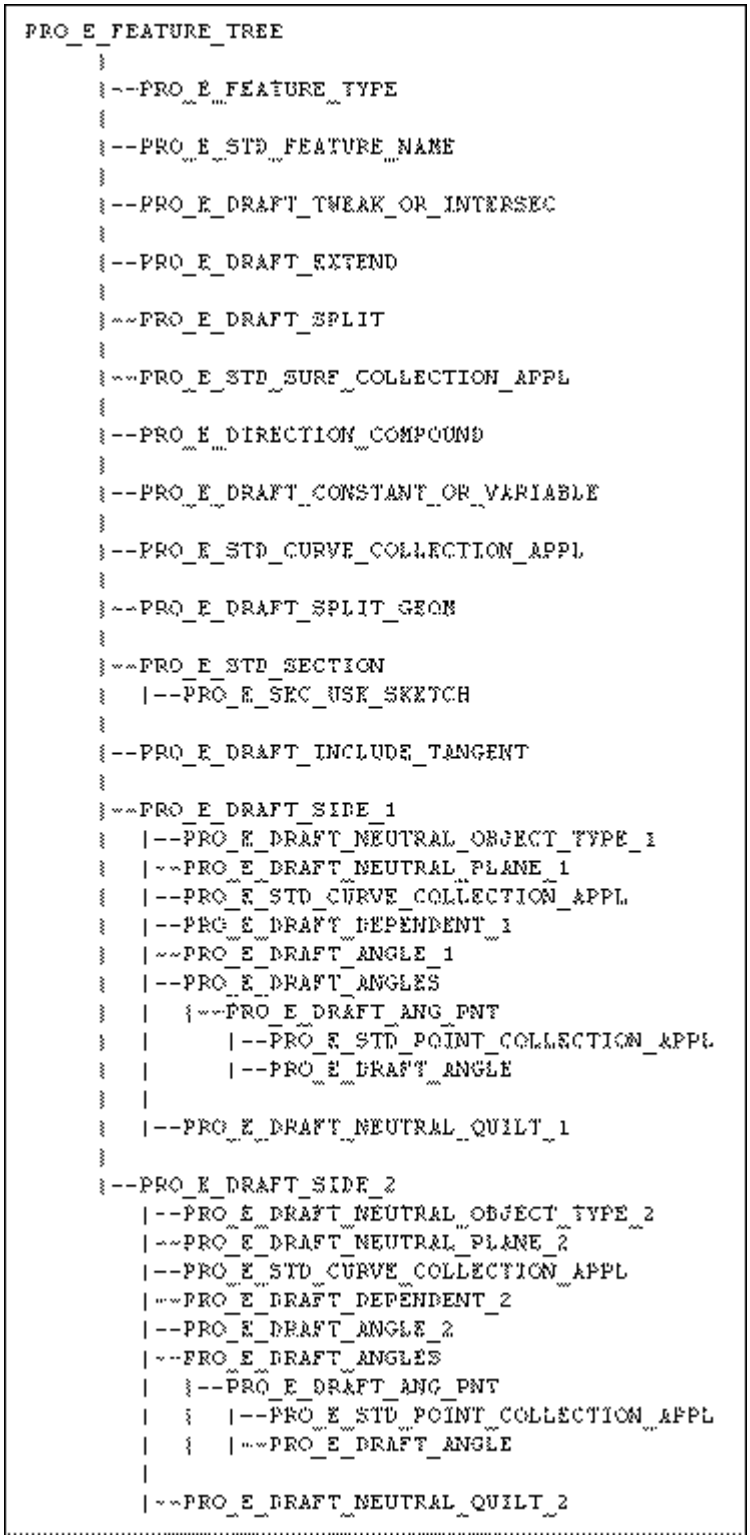
Some of the terms associated with the Draft feature are:

- Draft surfaces—The surfaces of the model to be drafted.
- Draft hinges—Lines and curves on the draft surfaces that the surfaces are pivoted about (also called neutral curves), or quilt of surfaces. Draft hinges can be defined by:
  - A plane, in which case the draft surfaces are pivoted about their intersection with the plane.
  - Individual curve chains on the draft surfaces.
  - A quilt, in which case the draft surfaces are pivoted about their intersection with the quilt.
- Draft direction—Direction used to measure the draft angle and can be defined in terms of:
  - A plane, in which case the draft direction is normal to this plane.
  - A straight edge or a datum axis, in which case the draft direction is parallel to the edge or axis.
  - Two points, such as datum points or model vertices, in which case the draft direction is parallel to the line connecting the two points.
  - A coordinate system, in which case the draft direction initially defaults to the direction of its x-axis.
- Draft angle—The angle between the draft direction and the resulting drafted surfaces. If the draft surfaces are split, you can define two independent angles for each side of the drafted surface. Draft angles must be within the range of -89.9° and +89.9°.

Feature Element Tree for the Draft Feature

The element tree for a draft feature is documented in the header file `ProDraft.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element tree for Draft Features



The feature element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- PRO_E_FEATURE_TYPE—Specifies the feature type.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.
- PRO_E_DRAFT_TWEAK_OR_INTERSEC—Specifies tweak or intersect depending on whether the resulting draft surface encounters an edge of the model or not suggesting the presence of the extend option. It can have any of the following values:
  - PRO_DRAFT_UI_TWEAK for creating regular draft geometry.
  - PRO_DRAFT_UI_INTERSECT for adjusting the draft geometry to intersect an existing edge of the model.
  - PRO_DRAFT_UI_INTERSECT_EXTEND specifies intersect with extend, when the draft does not extend to the adjacent model surface.

Note

It is an option for the earlier versions of Pro/ENGINEER.

- PRO_E_DRAFT_EXTEND—Specifies extend option of the draft. It is of the following types:
  - PRO_DRAFT_UI_NO_EXTEND—Intersect without Extend.
  - PRO_DRAFT_UI_EXTEND—Intersect with Extend.

Note

It is applicable for features created using Pro/ENGINEER version prior to Pro/ENGINEER Wildfire2.0 Release and is available only when PRO_E_DRAFT_TWEAK_OR_INTERSEC is equal to PRO_DRAFT_UI_INTERSECT.

- PRO_E_DRAFT_SPLIT—Specifies split details of the draft. It can be any of the following types:
  - PRO_DRAFT_UI_SPLIT_NONE
  - PRO_DRAFT_UI_SPLIT_NEUT specifies split on draft hinge.
  - PRO_DRAFT_UI_SPLIT_SURF specifies split at surface.
  - PRO_DRAFT_UI_SPLIT_SCTCH specifies split at sketch.

Note

Draft surfaces can be split either by the draft hinge or by a different curve on the draft surface, such as an intersection with a quilt, or a sketched curve. If you are splitting by a sketch that does not lie on the draft surface, Creo Parametric projects it on the draft surface in the direction normal to the sketching plane.

- PRO_E_STD_SURF_COLLECTION_APPL—Specifies drafted surfaces.
- PRO_E_DIRECTION_COMPOUND—Specifies the direction utility for the draft.
- PRO_E_DRAFT_CONSTANT_OR_VARIABLE—Specifies constant or variable draft. For variable draft one can specify more than one angle per draft side. It can be one of the following types:
  - PRO_DRAFT_UI_VARIABLE
  - PRO_DRAFT_UI_CONSTANT
- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the exclude loop or the draft hinges.
- PRO_E_DRAFT_SPLIT_GEOM—Specifies split geometry (sketch, plane, or quilt used for splitting) and is an optional element.
- PRO_E_STD_SECTION—Specifies the split geometry and is an optional element. It contains the following element:
  - PRO_E_SEC_USE_SKETCH—Specifies the selected split geometry and is an optional element.
- PRO_E_DRAFT_INCLUDE_TANGENT—Specifies included tangent. It can be any of the following types:
  - PRO_DRAFT_UI_NOT_INC_TANG specifies the non-included tangents.
  - PRO_DRAFT_UI_INC_TANG specifies the included tangents.
- PRO_E_DRAFT_SIDE_1—Specifies details about first draft's side.
- PRO_E_DRAFT_SIDE_2—Specifies details about second draft's side.

Element Details of PRO_E_DRAFT_SIDE_1

Each PRO_E_DRAFT_SIDE_1 has the following elements:

- PRO_E_DRAFT_NEUTRAL_OBJECT_TYPE_1—Specifies the type of draft hinge. It can be any of the following types:

- PRO_DRAFT_UI_NO_NEUT—Specifies that no draft hinge have been fixed.
- PRO_DRAFT_UI_PLANE—Specifies a plane. In this case, the draft surfaces are pivoted about their intersection with this plane.
- PRO_DRAFT_UI_CURVE—Specifies a curve chain located on the draft surfaces.
- PRO_DRAFT_UI_QUILT—Specifies a quilt of surfaces. In this case, the draft surfaces are pivoted about their intersection with the quilt.
- PRO_DRAFT_UI_RND_HINGE—Specifies a round surface that must be adjacent to the draft surface.

- PRO_E_DRAFT_NEUTRAL_PLANE_1—Specifies the plane selected as the draft hinge.
- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies exclude loop or draft hinges.
- PRO_E_DRAFT_DEPENDENT_1—Specifies the dependence and controls whether the corresponding sides are drafted and depends on the type of the draft hinge. It can be any of the following types:
  - PRO_DRAFT_UI_INDEPENDENT specifies that two independent draft angles for each side of the drafted surface.
  - PRO_DRAFT_UI_DEPENDENT specifies a single draft angle, with the second side drafted in the opposite direction.
  - PRO_DRAFT_UI_NONE specifies that none of the sides be drafted.
- PRO_E_DRAFT_ANGLE_1—Specifies the draft angle and is a constant value.
- PRO_E_DRAFT_ANGLES—This is an option for a variable draft. It specifies a collection of draft angles and points PRO_E_DRAFT_ANG_PNT. Each PRO_E_DRAFT_ANG_PNT consists of the following elements:
  - PRO_E_STD_POINT_COLLECTION_APPL—Specifies the point collection for the angle.
  - PRO_E_DRAFT_ANGLE—Specifies the draft angle.
- PRO_E_DRAFT_NEUTRAL_QUILT_1—Specifies the quilt of surfaces selected as the draft hinge.

Element Details of PRO_E_DRAFT_SIDE_2

Each PRO_E_DRAFT_SIDE_2 has the following elements:

- PRO_E_DRAFT_NEUTRAL_OBJECT_TYPE_2—Specifies the type of draft hinge. It can be any of the following types:

    - PRO_DRAFT_UI_NO_NEUT—Specifies that no draft hinge has been fixed.
    - PRO_DRAFT_UI_PLANE—Specifies a plane. In this case, the draft surfaces are pivoted about their intersection with this plane.
    - PRO_DRAFT_UI_CURVE—Specifies a curve chain located on the draft surfaces.
    - PRO_DRAFT_UI_QUILT—Specifies a quilt of surfaces. In this case, the draft surfaces are pivoted about their intersection with the quilt.
- PRO_E_DRAFT_NEUTRAL_PLANE_2—Specifies the plane selected as the draft hinge.
- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies exclude loop or draft hinges.
- PRO_E_DRAFT_DEPENDENT_2—Specifies the dependence and controls whether the corresponding sides are drafted and depends on the type of the draft hinge. It can be any of the following types:
    - PRO_DRAFT_UI_INDEPENDENT specifies that two independent draft angles for each side of the drafted surface.
    - PRO_DRAFT_UI_DEPENDENT specifies a single draft angle, with the second side drafted in the opposite direction.
    - PRO_DRAFT_UI_NONE specifies that none of the side be drafted.
- PRO_E_DRAFT_ANGLE_2—Specifies the draft angle and is a constant value.
- PRO_E_DRAFT_ANGLES—This is an option for a variable draft. It specifies a collection of draft angles and points PRO_E_DRAFT_ANG_PNT. Each PRO_E_DRAFT_ANG_PNT consists of the following elements:
    - PRO_E_STD_POINT_COLLECTION_APPL—Specifies the point collection for the angle.
    - PRO_E_DRAFT_ANGLE—Specifies the draft angle.
- PRO_E_DRAFT_NEUTRAL_QUILT_2—Specifies the quilt of surfaces selected as the draft hinge.

Creating a Draft

Function Introduced:

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Draft based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of the chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Draft

Function Introduced:

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Draft based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of the chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Draft

Function Introduced:

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Draft and to retrieve the element tree description of a Draft. For more information about `ProFeatureElemtreeExtract()` refer to the section Feature Inquiry on page 785 of the chapter Element Trees: Principles of Feature Creation on page 764.

Example 1: Creation of a Draft Feature

The sample code in the file `UgSimpleDraftCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a Simple Draft feature.

The user is prompted to select:

1. The surface to be drafted
2. The hinge (edge / curve)
3. The direction of the draft (axis, edge)

Example 2: Creation of a Draft Feature using interactive collection

The sample code in the file `UgIntcollectionDraftCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a draft feature using interactive collection.

The user is prompted to create a collection of:

1. The surface to be drafted ("ProSurfacesCollect")
2. The direction of the draft (axis, edge - "ProSelect")
3. The hinge (edge / curve - "ProCurvesCollect")

Example 3: Creation of a Draft Feature based on the Object-Action paradigm

The sample code in the file `UgOADraftCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` demonstrates creation of a simple Draft feature based on the Object-Action paradigm using selection buffer access. The user is expected to populate the selection buffer in Creo Parametric user interface with

1. A surface collection for the surfaces to be drafted
2. A curve collection for the hinge (edge / curve)
3. A selection for the direction of the draft (axis, edge)

This example parses the selection buffer and uses the information for the programmatic creation of draft feature. The selection buffer may be populated with the above three entries in any order. The success of feature creation depends upon the appropriateness of the data in the selection buffer.

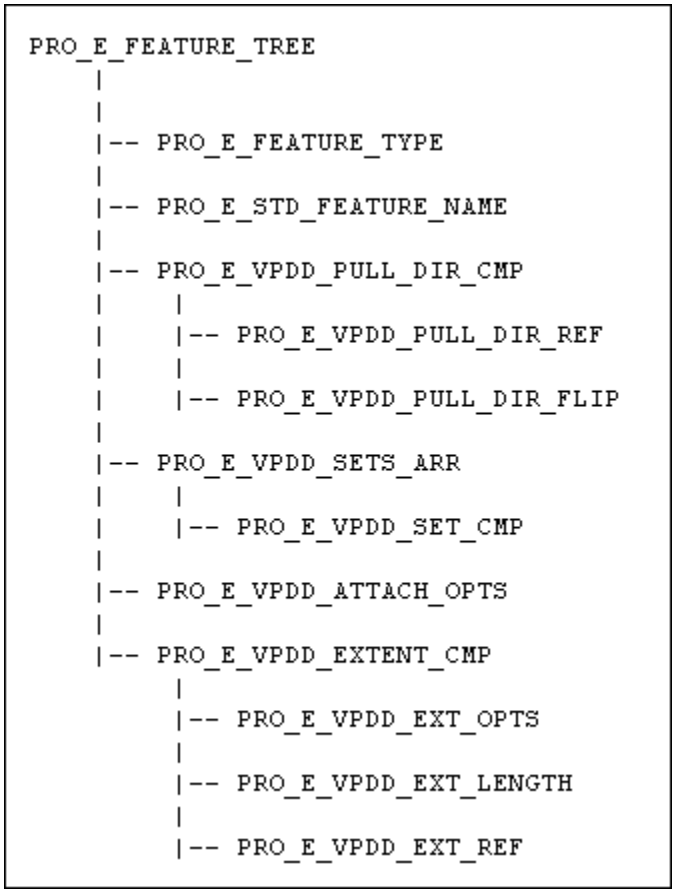
Variable Pull Direction Draft Feature

The Variable Pull Direction Draft (VPDD) feature differs from the regular Draft feature where the pull direction is restricted to be constant. A VPDD is defined by an edge or curve, a surface that specifies the variable pull direction, a depth option, and optionally, splitting surfaces.

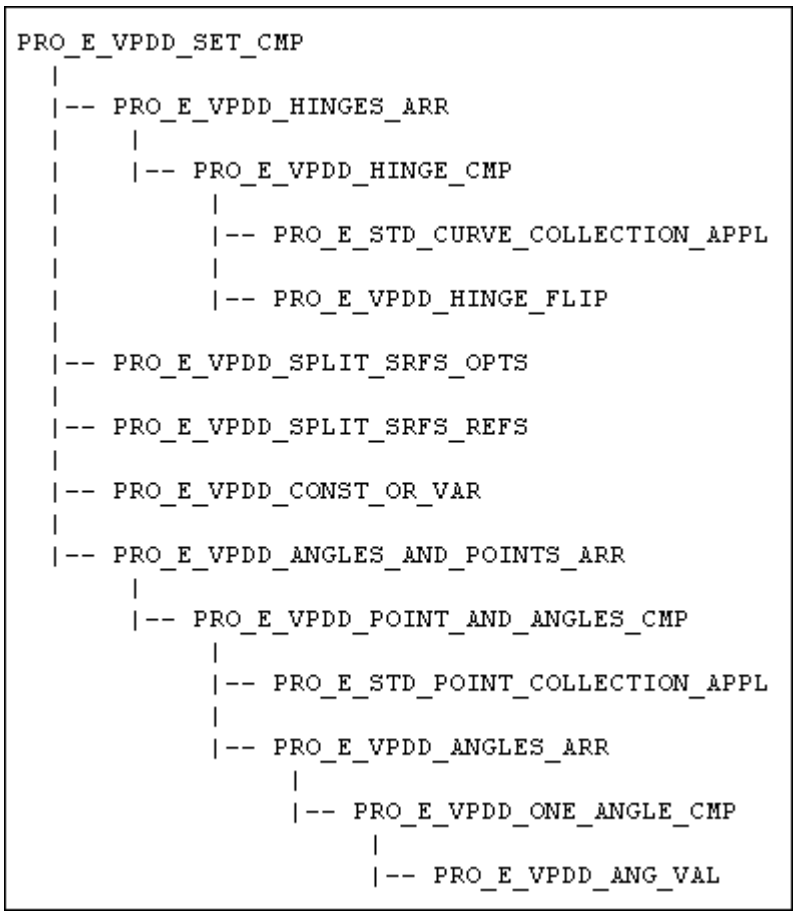
Feature Element Tree for the Variable Pull Direction Draft Feature

The element tree for the Variable Pull Direction Draft feature is documented in the header file `ProVPDD.h`. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Variable Pull Direction Draft Feature



PRO_E_VPDD_SET_CMP



The elements in this tree are as follows:

- PRO_E_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_VPDD.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.
- PRO_E_VPDD_PULL_DIR_CMP—Specifies the pull direction reference. This compound element consists of the following elements:
  - PRO_E_VPDD_PULL_DIR_REF—Specifies the selected reference surface. It could be a single PRO_QUILT, a single PRO_DATUM_PLANE, a single PRO_LOG_SRF, or multiple PRO_SURFACE references that are tangent to each other.
  - PRO_E_VPDD_PULL_DIR_FLIP—Specifies the pull direction defined by the normal vectors of the selected reference surfaces. By default, this element is PRO_B_TRUE and the pull direction is along the normal vectors. The pull direction can be flipped.

- PRO_E_VPDD_SETS_ARR—Specifies an array of the elements of the type PRO_E_VPDD_SET_CMP. PRO_E_VPDD_SET_CMP is a compound element and specifies a draft set. For more information on the elements contained by this compound element, refer to the section Element Details of the Subtree PRO_E_VPDD_SET_CMP on page 898.
- PRO_E_VPDD_ATTACH_OPTS—Specifies the attachment option. When all the draft hinges are two-sided edges, you can specify whether to attach the draft geometry to the existing solid or quilt, or create the draft geometry as a separate quilt. The attachment options, specified by the enumerated type Pro_vpdd_attach_type, are as follows:
  - PRO_VPDD_ATTACH_NEW_QUILT—Creates the draft geometry as a separate quilt.
  - PRO_VPDD_ATTACH_SAME_QUILT—Attaches the draft geometry to the existing solid or quilt.
- PRO_E_VPDD_EXTENT_CMP—Specifies the extent (depth) option when unattached draft geometry is created. This compound element is available only if the element PRO_E_VPDD_ATTACH_OPTS is set to PRO_VPDD_ATTACH_NEW_QUILT.
  - PRO_E_VPDD_EXT_OPTS—Specifies the extent options. These options, specified by the enumerated type Pro_vpdd_extent_type, are as follows:
    - PRO_VPDD_EXT_LENGTH—Specifies the length dimension option.
    - PRO_VPDD_EXT_TO_SEL—Select the reference bottom surface upto which the draft geometry is extended.
    - PRO_VPDD_EXT_TO_NEXT—The draft geometry is extended up to the next surface it intersects. This excludes any surface or quilt used as a parting surface for a different draft set.
    - PRO_VPDD_EXT_UNATTACHED—Creates the draft geometry ready for attachment but as a separate quilt.
    - PRO_E_VPDD_EXT_LENGTH—Specifies the length value. This element is available only when the element PRO_E_VPDD_EXT_OPTS is set to PRO_VPDD_EXT_LENGTH.
    - PRO_E_VPDD_EXT_REF—Specifies the selected bottom surface upto which the draft geometry is extended. The valid references are PRO_QUILT, PRO_SURFACE, and PRO_DATUM_PLANE. This element is available only when the element PRO_E_VPDD_EXT_OPTS is set to PRO_VPDD_EXT_TO_SEL.

Element Details of the Subtree PRO_E_VPDD_SET_CMP

The compound element PRO_E_VPDD_SET_CMP contains the following elements:

- PRO_E_VPDD_HINGES_ARR—Specifies an array of elements of the type PRO_E_VPDD_HINGE_CMP. PRO_E_VPDD_HINGE_CMP is a compound element and specifies a collection of draft hinges used to generate the draft geometry.
  - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies a chain of two-sided edges, one-sided edges, curves, or a combination of one-sided edges and curves.
  - PRO_E_VPDD_HINGE_FLIP—Specifies the ProBoolean option to select the sets of surfaces to be drafted on each side of a draft hinge defined by two-sided edges.
- PRO_E_VPDD_SPLIT_SRFS_OPTS—Specifies the ProBoolean option to specify if splitting surfaces are used while generating the draft geometry. If this option is set to the default value PRO_B_FALSE, then the element PRO_E_VPDD_SPLIT_SRFS_REFS becomes unavailable.
- PRO_E_VPDD_SPLIT_SRFS_REFS—Specifies a collection of splitting surface references. You can select upto two references that are either PRO_QUILT or PRO_DATUM_PLANE.
- PRO_E_VPDD_CONST_OR_VAR—Specifies a constant or variable draft. For a variable draft, there are variable angle attachment points. The values for this element, specified by the enumerated type Pro_vpdd_const_var_type, are as follows:
  - PRO_VPDD_CONST
  - PRO_VPDD_VAR
- PRO_E_VPDD_ANGLES_AND_POINTS_ARR—Specifies an array of elements of the type PRO_E_VPDD_POINT_AND_ANGLES_CMP. PRO_E_VPDD_POINT_AND_ANGLES_CMP is a compound element and specifies a collection of points and angles.
  - PRO_E_STD_POINT_COLLECTION_APPL—Specifies the selection references (curves, edges, or datum points) where the draft angles are defined.

Note

- If PRO_E_VPDD_CONST_OR_VAR is set to PRO_VPDD_CONST, then the element PRO_E_STD_POINT_COLLECTION_APPL becomes unavailable and the element PRO_E_VPDD_ANGLES_AND_POINTS_ARR contains only a single element.
- A reference is valid when it belongs to or lies on a draft hinge that belongs to the same draft set.

- PRO_E_VPDD_ANGLES_ARR—Specifies an array of elements of the type PRO_E_VPDD_ONE_ANGLE_CMP. The size of the array is one plus the number of splitting references. PRO_E_VPDD_ONE_ANGLE_CMP is a compound element and specifies the draft angle for the active draft set.
  - PRO_E_VPDD_ANG_VAL—Specifies the value of the draft angle. This value lies in the range of -90 degrees through 90 degrees.

Creating a VPDD

Function Introduced:

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a VPDD based on element tree input. For more information about ProFeatureCreate(), refer to the section Overview of Feature Creation on page 765 of the chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a VPDD

Function Introduced:

- **ProFeatureRedefine()**

Use the function ProFeatureRedefine() to redefine a VPDD based on the changes made in the element tree. For more information about ProFeatureRedefine(), refer to the section Feature Redefine on page 786 of the chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a VPDD

Function Introduced:

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a VPDD and to retrieve the element tree description of a Draft. For more information about `ProFeatureElemtreeExtract()` refer to the section Feature Inquiry on page 785 of the chapter Element Trees: Principles of Feature Creation on page 764.

39

Element Trees: Round and Chamfer

This chapter describes how to create, redefine, and query round and chamfer features through element trees and element tree functions. The section Overview of Feature Creation on page 765 of the chapter Element Trees: Principles of Feature Creation on page 764 provides a necessary background information for this topic.

See the Part Modeling module in Creo Parametric for further details on round and chamfer feature creation.

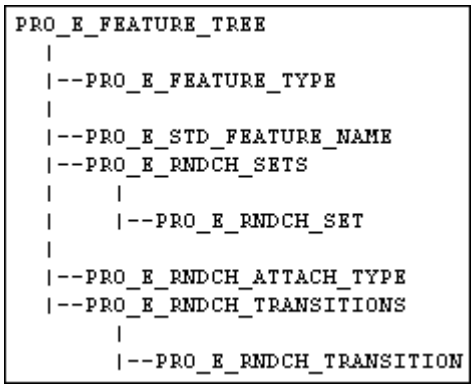
The following sections describe the procedure for accessing the chamfers and round features in detail.

Round Feature

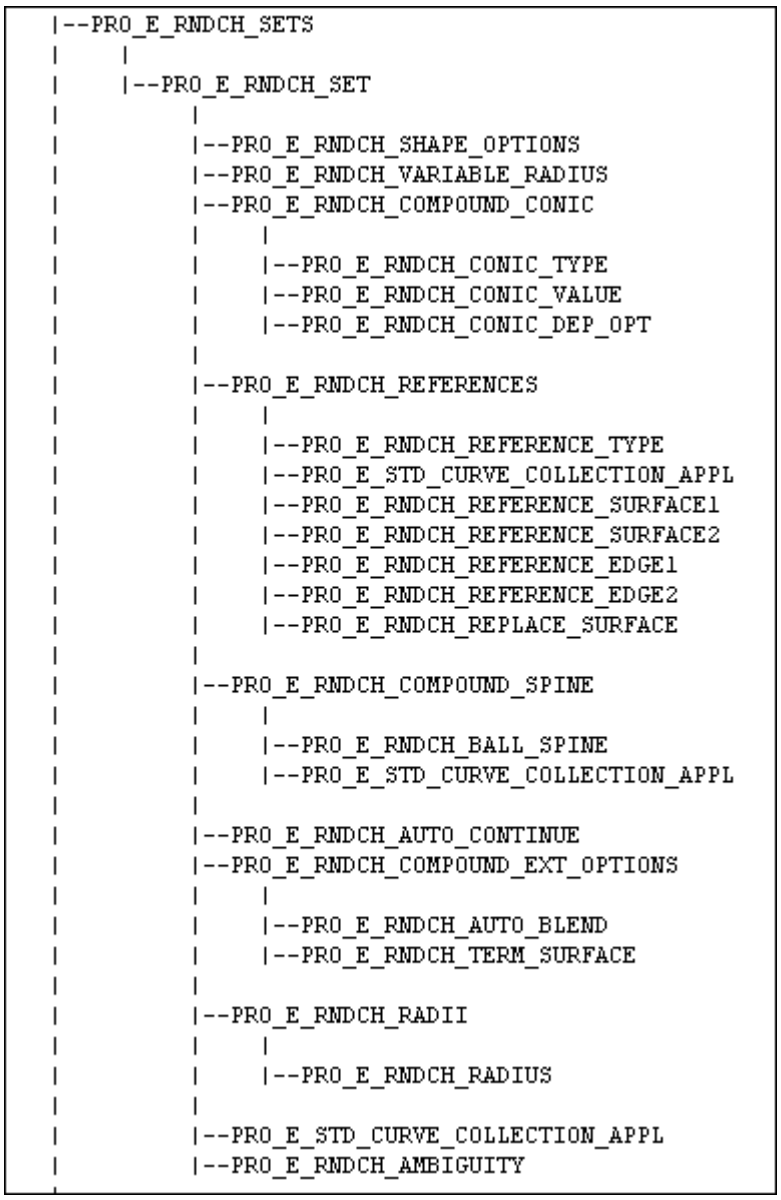
Feature Element Tree for Round Feature

The element tree for a round is documented in the header file `ProRound.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element Tree for Round



PRO_E_RNDCH_SETS



PRO_E_RNDCH_RADIUS

```
--PRO_E_RNDCH_RADIUS
    |
    |--PRO_E_STD_POINT_COLLECTION_APPL
    |--PRO_E_RNDCH_LEG1
    |     |
    |     |--PRO_E_RNDCH_LEG_TYPE
    |     |--PRO_E_RNDCH_LEG_VALUE
    |     |--PRO_E_RNDCH_REFERENCE_EDGE
    |     |--PRO_E_RNDCH_REFERENCE_POINT
    |
    |--PRO_E_RNDCH_LEG2
          |
          |--PRO_E_RNDCH_LEG_TYPE
          |--PRO_E_RNDCH_LEG_VALUE
          |--PRO_E_RNDCH_REFERENCE_EDGE
          |--PRO_E_RNDCH_REFERENCE_POINT
```

PRO_E_RNDCH_TRANSITIONS

```
|--PRO_E_RNDCH_TRANSITIONS
     |
     |--PRO_E_RNDCH_TRANSITION
           |
           |--PRO_E_RNDCH_TRANS_TYPE
           |--PRO_E_RNDCH_TRANS_CAP
           |--PRO_E_RNDCH_TRANS_SPHERE_DATA
           |     |
           |     |--PRO_E_RNDCH_TRANS_RADIUS_OPTIONS
           |     |--PRO_E_RNDCH_TRANS_SPHERE_RADIUS
           |     |--PRO_E_RNDCH_TRANS_LEG1_OPTIONS
           |     |--PRO_E_RNDCH_TRANS_LEG1_VALUE
           |     |--PRO_E_RNDCH_TRANS_LEG2_OPTIONS
           |     |--PRO_E_RNDCH_TRANS_LEG2_VALUE
           |     |--PRO_E_RNDCH_TRANS_LEG3_OPTIONS
           |     |--PRO_E_RNDCH_TRANS_LEG3_VALUE
           |
           |--PRO_E_RNDCH_TRANS_PATCH_DATA
           |     |
           |     |--PRO_E_RNDCH_TRANS_PATCH_REF_SURF
           |     |--PRO_E_RNDCH_TRANS_PATCH_RAD_OPT
           |     |--PRO_E_RNDCH_TRANS_ARC_RADIUS
           |
           |--PRO_E_RNDCH_TRANS_STOP_DATA
                 |
                 |--PRO_E_RNDCH_TRANS_STOP_REF_TYPE
                 |--PRO_E_RNDCH_TRANS_STOP_REFERENCE
```

The following list details special information about the elements in this tree:

- PRO_E_FEATURE_TYPE—Specifies feature type and should have the value as PRO_FEAT_ROUND only.
- PRO_E_STD_FEATURE_NAME—Specifies the name to the feature. This is an optional element. If not specified then a default name will be assigned internally to the feature.
- PRO_E_RNDCH_SETS—Specifies an array of PRO_E_RNDCH_SET.
- PRO_E_RNDCH_ATTACH_TYPE—Specifies the attachment type and has the following values:
  - PRO_ROUND_ATTACHED—Specifying this option, the created round feature consumes the model geometry.
  - PRO_ROUND_UNATTACHED
  - PRO_ROUND_CAPPED_ENDS
- PRO_E_RNDCH_TRANSITIONS—Specifies a set of transition PRO_E_RNDCH_TRANSITION.

Element Details of PRO_E_RNDCH_SET for Round

Each PRO_E_RNDCH_SET specifies a round set, which is a round piece (geometry) created as per the placement references and consists of the following elements:

- PRO_E_RNDCH_SHAPE_OPTIONS—Specifies the shape options and have the following values:
  - PRO_ROUND_TYPE_CONSTANT—Specifies a round piece having a constant radius.
  - PRO_ROUND_TYPE_VARIABLE—Specifies a round piece having multiple radii.
  - PRO_ROUND_TYPE_FULL—Specifies full round which replaces the selected surface.
  - PRO_ROUND_TYPE_THROUGH_CURVE—Allows the radius of the active round to be driven by the selected datum curve.
- PRO_E_RNDCH_VARIABLE_RADIUS—Specifies if the round is of a constant or variable type.
- PRO_E_RNDCH_COMPOUND_CONIC—Specifies if the round uses a conic section for the shape. It can be defined as:
  - PRO_E_RNDCH_CONIC_TYPE—Specifies conic type and can have the following valid values:
    - PRO_ROUND_CONIC_ENABLE

    - PRO_ROUND_CONIC_DISABLE
    - PRO_ROUND_CURV_CONTINUOUS—Used to specify curvature continuous rounds.
  - PRO_E_RNDCH_CONIC_VALUE—Specifies conic value, that controls the sharpness of the conic shape and is required if PRO_E_RNDCH_CONIC_TYPE is equal to PRO_ROUND_CONIC_ENABLE.
  - PRO_E_RNDCH_CONIC_DEP_OPT—Specifies independent options and is required if PRO_E_RNDCH_CONIC_TYPE is equal to PRO_ROUND_CONIC_ENABLE.
  - The values of ProRoundConicIndependentType are as follows:
    - PRO_ROUND_CONIC_DEPENDENT
    - PRO_ROUND_CONIC_INDEPENDENT
    - PRO_ROUND_CONIC_INDEPENDENT is the default type.
- PRO_E_RNDCH_REFERENCES—Specifies a set of valid references of the round feature.
- PRO_E_RNDCH_COMPOUND_SPINE—This is another option for defining the shape of the round. Specifies the spine and has the following elements:
  - PRO_E_RNDCH_BALL_SPINE—Specifies rolling ball or normal to spine. Valid values of PRO_E_RNDCH_BALL_SPINE are:
    - PRO_ROUND_ROLLING_BALL—Specifies a round surface and is created by rolling a spherical ball along the surfaces.
    - PRO_ROUND_NORMAL_TO_SPINE—Specifies a round surface and is created by sweeping a conic or arc cross-section normal to a spine.
    - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the spine curve and is required if PRO_E_RNDCH_BALL_SPINE is equal to PRO_ROUND_NORMAL_TO_SPINE.

Note

During the creation of rounds, the options **D1 x D2 Conic** and **Normal to spine** cannot be used together. Due to this restriction, the existing rounds with their conic type option set as PRO_ROUND_CONIC_INDEPENDENT and with the round creation method set to PRO_ROUND_NORMAL_TO_SPINE are reset to PRO_ROUND_ROLLING_BALL when the round is redefined. Therefor, for the conic type option PRO_ROUND_CONIC_INDEPENDENT you must specify the round creation method as PRO_ROUND_ROLLING_BALL.

- PRO_E_RNDCH_AUTO_CONTINUE—Specifies whether surfaces will be extended to meet the designated round radius. The valid values are:
  - PRO_ROUND_AUTO_CONT_DISABLE—This is the default value.
  - PRO_ROUND_AUTO_CONT_ENABLE

  This element is required if PRO_E_RNDCH_REFERENCE_TYPE = PRO_ROUND_REF_EDGE.
- PRO_E_RNDCH_COMPOUND_EXT_OPTIONS—Specifies the external options. This is an optional element. It has the following elements:
  - PRO_E_RNDCH_AUTO_BLEND—Specifies auto blend.
  - PRO_E_RNDCH_TERM_SURFACE—Specifies the terminating surface.
- PRO_E_RNDCH_RADII—Specifies the radii, as an array of radius or PRO_E_RNDCH_RADIUS and is required if PRO_E_RNDCH_SHAPE_OPTIONS is not equal to PRO_ROUND_TYPE_THROUGH_CURVE and PRO_E_RNDCH_SHAPE_OPTIONS is not equal to PRO_ROUND_TYPE_FULL.
- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the curve collection and is required if PRO_E_RNDCH_SHAPE_OPTIONS is equal to PRO_ROUND_TYPE_THROUGH_CURVE.
- PRO_E_RNDCH_AMBIGUITY—Specifies round set ambiguity.

Note

Ambiguity occurs in round features when other placement locations exist for the round set. The ambiguity condition occurs when two surfaces intersect in multiple locations.

Element Details of PRO_E_RNDCH_REFERENCES for Round

Each PRO_E_RNDCH_REFERENCES specifies a set of valid references of the round feature and has the following elements:

- PRO_E_RNDCH_REFERENCE_TYPE—Specifies the reference types and valid values are:
  - PRO_ROUND_REF_EDGE—Specifies that the surfaces border the edge reference and form the rolling tangent attachment for the round.
  - PRO_ROUND_REF_SURF_SURF—Specifies that the edges of the round remain tangent to the reference surfaces.

- PRO_ROUND_REF_EDGE_SURF—Specifies that the round remains tangent to the surface. The edge reference does not maintain tangency.
- PRO_ROUND_REF_EDGE_EDGE—Specifies that the surfaces bordering the edge reference form the rolling tangent attachment for the round.

- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the reference edges of the chain collection. It is required if PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_EDGE.

  In Creo Parametric TOOLKIT 7.0.0.0 and later, you can select the reference edges from both different solid bodies as well as quilts. The resulting geometry is attached back to the same solid body or quilt from where the referenced edges were selected.
- PRO_E_RNDCH_REFERENCE_SURFACE1—Specifies the first reference surface and is required if PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_SURF_SURF.
- PRO_E_RNDCH_REFERENCE_SURFACE2—Specifies the second reference surface and is required if PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_SURF_SURF or PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_EDGE_SURF.
- PRO_E_RNDCH_REFERENCE_EDGE1—Specifies the first reference edge and is required if PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_EDGE_SURF or PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_EDGE_EDGE and PRO_E_RNDCH_SHAPE_OPTIONS is equal to PRO_ROUND_TYPE_FULL.
- PRO_E_RNDCH_REFERENCE_EDGE2—Specifies the second reference edge and is required if PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_EDGE_EDGE and PRO_E_RNDCH_SHAPE_OPTIONS is equal to PRO_ROUND_TYPE_FULL.
- PRO_E_RNDCH_REPLACE_SURFACE—Specifies the surface to be replaced and is required if PRO_E_RNDCH_REFERENCE_TYPE is equal to PRO_ROUND_REF_SURF_SURF and PRO_E_RNDCH_SHAPE_OPTIONS is equal to PRO_ROUND_TYPE_FULL.

Element Details of PRO_E_RNDCH_RADIUS for Round

Each PRO_E_RNDCH_RADIUS has the following elements:

- PRO_E_STD_POINT_COLLECTION_APPL—Specifies a point, which governs the value of the radius.
- PRO_E_RNDCH_LEG1—Specifies the leg1.
- PRO_E_RNDCH_LEG2—Specifies the leg2 and is required if PRO_E_RNDCH_CONIC_DEP_OPT is equal to PRO_ROUND_CONIC_INDEPENDENT.

Each PRO_E_RNDCH_LEG1 or PRO_E_RNDCH_LEG2 has the following elements:

- PRO_E_RNDCH_LEG_TYPE—ProRoundRadiusType specifies leg type and is a mandatory element. It is of the following types:
  - PRO_ROUND_RADIUS_TYPE_VALUE
  - PRO_ROUND_RADIUS_THROUGH_POINT
- PRO_E_RNDCH_LEG_VALUE—Specifies leg value and is required if PRO_E_RNDCH_LEG_TYPE is equal to PRO_ROUND_RADIUS_TYPE_VALUE.
- PRO_E_RNDCH_REFERENCE_EDGE—Specifies reference edge having the value as PRO_E_EDGE and is required if PRO_E_RNDCH_LEG_TYPE is equal to PRO_ROUND_RADIUS_THROUGH_POINT.
- PRO_E_RNDCH_REFERENCE_POINT—Specifies reference point having the value as PRO_E_POINT and is required if PRO_E_RNDCH_LEG_TYPE is equal to PRO_ROUND_RADIUS_THROUGH_POINT.

Element Details of PRO_E_RNDCH_TRANSITION for Round

Each PRO_E_RNDCH_TRANSITION represents user-defined transitions for the entire round feature and consists of the following elements:

- PRO_E_RNDCH_TRANS_TYPE specifies the type of the transition type. Valid values are:
  - PRO_ROUND_TRANS_INTERSECT—Used to extend two or more overlapping round pieces towards each other until they merge forming a sharp boundary.
  - PRO_ROUND_TRANS_BLEND—Used to create a fillet surface between the round pieces using an edge reference.
  - PRO_ROUND_TRANS_STOP—Used to terminate the round geometry at the specified datum point or datum plane.
  - PRO_ROUND_TRANS_CONTINUE—Used to extend round geometry into two round pieces.

- PRO_ROUND_TRANS_SPHERE_CORNER—Used to create a round from the corner transition formed by three overlapping pieces by a spherical corner.
- PRO_ROUND_TRANS_PATCH—Used to create a patched surface at the location where three or four round pieces overlap at a corner.
- PRO_ROUND_TRANS_BLEND_3SRF—Used to create a triangular patch as a transition of three rounds.
- PRO_ROUND_TRANS_RBALL—Used to create a rolling ball transition for three or more rounds.
- PRO_ROUND_TRANS_STOP_0_SIDE—Used to terminate the round using geometry configured by Creo Parametric.
- PRO_ROUND_TRANS_STOP_1_SIDE—Used to terminate the round using geometry configured by Creo Parametric.
- PRO_ROUND_TRANS_STOP_2_SIDE—Used to terminate the round using geometry configured by Creo Parametric.
- PRO_ROUND_TRANS_STOP_AT_REF—Used to terminate round geometry at the selected datum point or datum plane.
- PRO_ROUND_TRANS_STOP_FULL—Used to keep the stop transition close to the boundary of the removed surface.
- PRO_ROUND_TRANS_STOP_2_WE—Used to create a stop transition with maximum possible extension of the round by freezing one of round's references and changing the other reference at intersection with tangent edges.

- PRO_E_RNDCH_TRANS_CAP—Specifies the capping surface for round feature. It has the following values:
  - PRO_ROUND_CAPPING_SURF_DISABLE = PRO_B_FALSE
  - PRO_ROUND_CAPPING_SURF_ENABLE = PRO_B_TRUE
- PRO_E_RNDCH_TRANS_SPHERE_DATA—Specifies sphere data and consists of the following elements:
  - PRO_E_RNDCH_TRANS_RADIUS_OPTIONS—Specifies radius options and is a mandatory element.
  - PRO_E_RNDCH_TRANS_SPHERE_RADIUS—Specifies sphere radius and is required if PRO_E_RNDCH_TRANS_RADIUS_OPTIONS is equal to PRO_ROUND_TRANS_RADIUS_ENTER_VALUE.
  - PRO_E_RNDCH_TRANS_LEG1_OPTIONS—Specifies leg1 options and is a mandatory element.

- PRO_E_RNDCH_TRANS_LEG1_VALUE—Specifies the value of leg1 and is required if PRO_E_RNDCH_TRANS_LEG1_OPTIONS is equal to PRO_ROUND_TRANS_RADIUS_ENTER_VALUE.
- PRO_E_RNDCH_TRANS_LEG2_OPTIONS—Specifies leg2 options and is a mandatory element.
- PRO_E_RNDCH_TRANS_LEG2_VALUE—Specifies the value of leg2 and is required if PRO_E_RNDCH_TRANS_LEG2_OPTIONS is equal to PRO_ROUND_TRANS_RADIUS_ENTER_VALUE.
- PRO_E_RNDCH_TRANS_LEG3_OPTIONS—Specifies leg3 options and is a mandatory element.
- PRO_E_RNDCH_TRANS_LEG3_VALUE—Specifies the value of leg3 and is required if PRO_E_RNDCH_TRANS_LEG3_OPTIONS is equal to PRO_ROUND_TRANS_RADIUS_ENTER_VALUE.

- PRO_E_RNDCH_TRANS_PATCH_DATA—Specifies the patch data and is required if PRO_E_RNDCH_TRANS_TYPE is equal to PRO_ROUND_TRANS_PATCH. It has the following elements:
  - PRO_E_RNDCH_TRANS_PATCH_REF_SURF—Specifies the arc surface, which indicates that a valid surface reference has been selected to place a fillet for the active patch transition. It should have the value as PRO_SURFACE only.
  - PRO_E_RNDCH_TRANS_PATCH_RAD_OPT—Specifies the arc radius options. It indicates the fillet radius for the active patch transition and has following options:
    - PRO_ROUND_TRANS_RADIUS_ENTER_VALUE— Specifies a new radius value.
    - PRO_ROUND_TRANS_RADIUS_AUTOMATIC— Specifies the most recently used radius value.
    - PRO_E_RNDCH_TRANS_ARC_RADIUS—Specifies the arc radius.
- PRO_E_RNDCH_TRANS_STOP_DATA—Specifies the capping surface. It has the following elements:
  - PRO_E_RNDCH_TRANS_STOP_REF_TYPE—Specifies the reference type. Valid values are:
    - PRO_ROUND_TRANS_REF_NO_REF
  - PRO_ROUND_TRANS_REF_GEOM
  - PRO_ROUND_TRANS_REF_PNTVTX
  - PRO_ROUND_TRANS_REF_DTMPLN
  - PRO_ROUND_TRANS_REF_ISOLINE

- PRO_E_RNDCH_TRANS_STOP_REFERENCE—Specifies the references for the active stop at reference transition. It can either be PRO_SURFACE or PRO_POINT.

Creating a Round

Function Introduced:

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a round based on element tree input. For more information about ProFeatureCreate(), refer to the section Overview of Feature Creation on page 765 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Note

In Pro/ENGINEER Wildfire 2.0,

- Pro/TOOLKIT does not support the temporary geometry required for user-specified ambiguity and non-default transitions. Therefore, these elements cannot be used for creation of new rounds.
- If transitions are specified in the input element tree, a round feature with the default transition will be created.
- In case of ambiguous situation (where more than one valid solutions exist, e.g. for surface-surface round - surfaces having discontinuous edges of intersection), a round feature with default solution will be created.

Redefining a Round

Function Introduced:

- **ProFeatureRedefine()**

Use the function ProFeatureRedefine() to redefine a round based on the changes made in the element tree. For more information about ProFeatureRedefine(), refer to the section Feature Redefine on page 786 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Note

In Pro/ENGINEER Wildfire 2.0,

- A round feature having default transition, can not be redefined to have any transition.
- A round feature having a Pro/ENGINEER user interface defined transition can be redefined to other type of transition, for example, from intersect type to spherical type. The input element tree must have a valid transition of the required type.

Accessing a Round

Function Introduced:

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a round and to retrieve the element tree description of a round. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Example 1: Sample code for creation of a Round Feature

The sample code in `UgEdgeRoundCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an Edge Round feature. The user is prompted for three edges to be rounded.

Modify Round Radius Feature

As a part of the flexible modeling capabilities the `Modify Round Radius` feature allows you to modify the radius of existing round geometry. You can modify the radius of both constant and variable rounds, however variable radius rounds are converted to constant radius rounds upon modification. You can also modify the radii of multiple round shape sets.

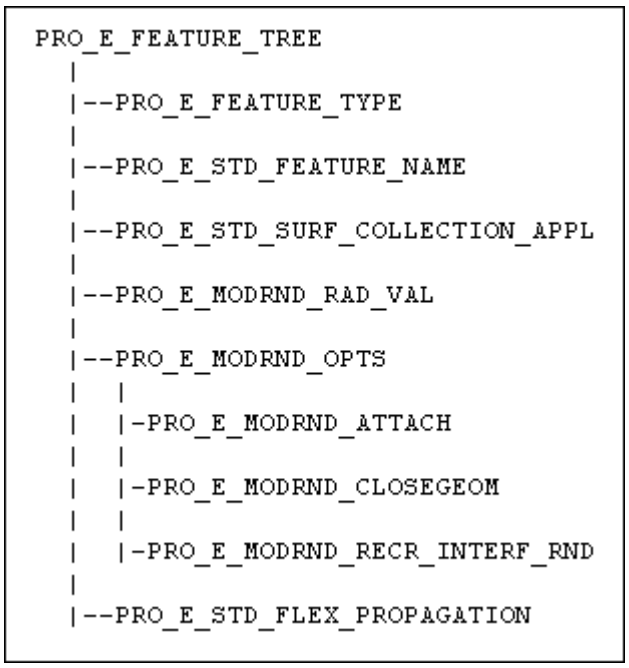
The creation of the `Modify Round Radius` feature includes the following steps:

1. Identify the round geometry to be modified and its radius value.
2. Creo Parametric removes the identified round geometry using the remove surface algorithm and recreates the rounded edges with the desired radius value.
3. The feature IDs of recreated rounds are updated.

Feature Element Tree for Modify Round Radius Feature

The element tree for a `Modify Round Radius` feature is documented in the header file `ProModifyRound.h` and has a simple structure. The following figure demonstrates the structure of the feature element tree.

Modify Round Radius Element Tree



The elements in this tree are described as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_MOD_ROUND`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature. The default value of this element is `MODIFY_ROUND`.
- `PRO_E_STD_SURF_COLLECTION_APPL`—Specifies a collection of reference surfaces the includes the rounds to be modified.

In Creo Parametric TOOLKIT 7.0.0.0 and later, you can select the reference surfaces from both different solid bodies as well as quilts. The resulting geometry is attached back to the same solid body or quilt from where the referenced surfaces were selected.

- PRO_E_MODRND_RAD_VAL—Specifies the new radius value. This value falls in the range [(part epsilon / 10.0), 1.0e+06].
- PRO_E_MODRND_OPTS—Specifies the modification options for each round to be recreated. This compound element consists of the following elements:
  - PRO_E_MODRND_ATTACH—Specifies whether the recreated round geometry is attached to the selected reference surface upon creation. The values for this element, specified by the enumerated type ProModRndAttach, are as follows:
    - PRO_MODRND_ATTACH_GEOM
    - PRO_MODRND_DONOT_ATTACH_GEOM
  - PRO_E_MODRND_CLOSEGEOM—Specifies if end surfaces are created for the recreated rounds. The values for this element, specified by the enumerated type ProModRndCloseGeom, are as follows:
    - PRO_MODRND_CLOSE_GEOM
    - PRO_MODRND_DONOT_CLOSE_GEOM

    Note

    The element PRO_E_MODRND_CLOSEGEOM is applicable only if the element PRO_E_MODRND_ATTACH has the value PRO_MODRND_DONOT_ATTACH_GEOM, meaning that the rounds are recreated as new surfaces or quilts.

  - PRO_E_MODRND_RECR_INTERF_RND—Specifies whether interfering rounds are removed and recreated in order to recreate the modified round geometry. The values for this element, specified by the enumerated type ProModRndRecrRounds, are as follows:
    - PRO_MODRND_RECR_INTERF_RNDS
    - PRO_MODRND_DONOT_RECR_INTERF_RNDS
- PRO_E_STD_FLEX_PROPAGATION—Specifies a Pattern feature, a Symmetry Recognition feature, or a Mirror Geometry feature that contains the reference surfaces specified by the element PRO_E_STD_SURF_COLLECTION_APPL. If such surfaces exist, then the modification of the round radius is propagated to all corresponding surfaces in the instances of the

Pattern feature, the Symmetry Recognition feature, or the Mirror Geometry feature, in order to maintain the pattern or symmetry.

Auto Round Feature

The Auto Round feature enables you to create round geometry of a constant radius on solid geometry or on a quilt of a part or assembly. The Auto Round Feature creates Round features called Auto-Round Members (ARMs) that are represented in the Model Tree as subnodes of the Auto Round feature, or as groups of individual independent round features. Refer to the Creo Parametric Online Help for more details about this feature type.

Pro/TOOLKIT does not provide access to the Auto Round feature via an element tree for the Wildfire 4.0 release. Some Creo Parametric TOOLKIT functionality such as feature deletion, suppression and redefinition are not supported for individual ARMs within an Auto Round feature.

Functions Introduced:

- **ProRoundIsAutoRoundMember()**

The function `ProRoundIsAutoRoundMember()` identifies if the specified round feature is a member of the Auto Round feature.

Chamfer Feature

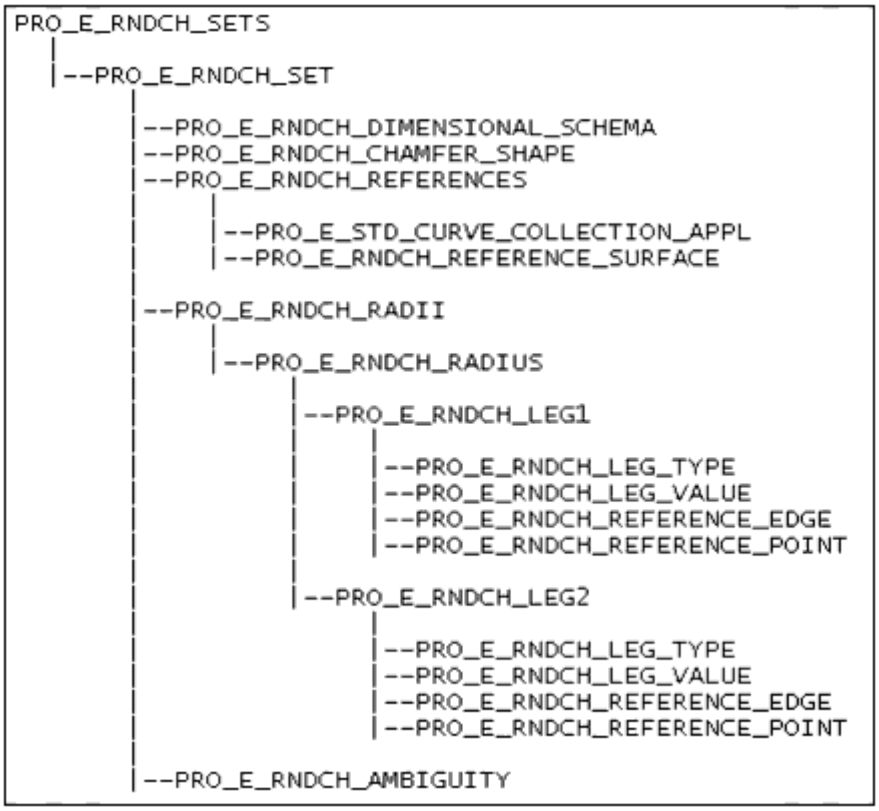
Feature Element Tree for Chamfer Feature

The element tree for a chamfer is documented in the header file `ProChamfer.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

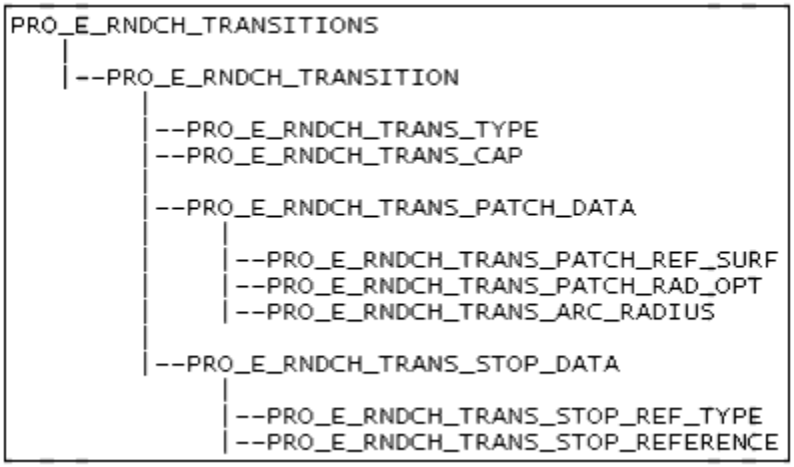
Feature Element Tree for Chamfer

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_STD_FEATURE_NAME
   |--PRO_E_RNDCH_SETS
   |     |
   |     |--PRO_E_RNDCH_SET
   |
   |--PRO_E_RNDCH_ATTACH_TYPE
   |--PRO_E_RNDCH_TRANSITIONS
         |
         |--PRO_E_RNDCH_TRANSITION
```

PRO_E_RNDCH_SETS



PRO_E_RNDCH_TRANSITIONS



The following list details special information about some of the elements in this tree:

- PRO_E_FEATURE_TYPE—Specifies feature type and has the value of PRO_FEAT_CHAMFER.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.
- PRO_E_RNDCH_SETS—Specifies an array of PRO_E_RNDCH_SET.
- PRO_E_RNDCH_ATTACH_TYPE—Specifies the attachment type and has the following values:
  - PRO_ROUND_ATTACHED
  - PRO_ROUND_UNATTACHED
  - PRO_ROUND_CAPPED_ENDS
- PRO_E_RNDCH_TRANSITIONS—Specifies a set of transition PRO_E_RNDCH_TRANSITION.

Element Details of PRO_E_RNDCH_SET for Chamfer

Each PRO_E_RNDCH_SET specifies a chamfer set for the chamfer feature and must have the following elements:

- PRO_E_RNDCH_DIMENSIONAL_SCHEMA—Specifies the type of chamfer or the dimensional schema PRO_E_RNDCH_DIMENSIONAL_SCHEMA using the enumerated type ProChmSchema. The different types of PRO_E_RNDCH_DIMENSIONAL_SCHEMA is as follows:
  - PRO_CHM_45_X_D—Specifies a chamfer that is at an angle of 45 degrees to both surfaces and at a distance D from the edge along each surface.
  - PRO_CHM_D_X_D—Specifies a chamfer that is at a distance D from the edge along each surface. This is the default type.
  - PRO_CHM_D1_X_D2—Specifies a chamfer at a distance D1 from the selected edge along one surface and at a distance D2 from the selected edge along the other surface.
  - PRO_CHM_ANG_X_D—Specifies a chamfer at a distance D from the selected edge along one adjacent surface, at a specified angle to that surface.
  - PRO_CHM_O_X_O—Provides direct control of the surface offset distances.
  - PRO_CHM_O1_X_O2—Provides direct control of the surface offset distances.

Note

For Surf-Surf chamfer, the available schemes are:

- Offset Surface method—PRO_CHM_O_X_O, PRO_CHM_O1_X_O2, PRO_CHM_D_X_D, PRO_CHM_D1_X_D2 are available.
- Tangent Dist method—PRO_CHM_D_X_D, PRO_CHM_D1_X_D2, PRO_CHM_45_X_D, PRO_CHM_ANG_X_D are available.

For Surface-to-Edge chamfer the available schemes are:

- Offset Surface: PRO_CHM_O_X_O, PRO_CHM_O1_X_O2.
- Tangent Distance: PRO_CHM_D_X_D, PRO_CHM_D1_X_D2.

These schemes are applicable for constant angle planes or constant 90 degree surfaces and are also available if all members of the edge chain are formed by exactly 2 planes or exactly 2 surfaces at 90 degree, as in the ends of a cylinder.

- PRO_E_RNDCH_CHAMFER_SHAPE—Specifies the shape of the chamfer feature. PRO_E_RNDCH_CHAMFER_SHAPE has the following valid values:
  - PRO_CHM_TANGENT_LEGS—Specifies the chamfer distance between vectors that are tangent to the neighboring surface of the reference edge.
  - PRO_CHM_OFFSET_SURFACE—Specifies the offset surfaces.
- PRO_E_RNDCH_REFERENCES—Specifies a set of valid references of the chamfer feature and has the following elements:
  - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies reference edges and is a required element for edge chamfer.

    In Creo Parametric TOOLKIT 7.0.0.0 and later, you can select the reference edge chamfers from both different solid bodies as well as quilts. The resulting geometry is attached back to the same solid body or quilt from where the referenced edges were selected.
  - PRO_E_RNDCH_REFERENCE_SURFACE—Specifies reference surfaces and is required if either PRO_E_RNDCH_DIMENSIONAL_SCHEMA is equal to PRO_CHM_D1_X_D2 or PRO_E_RNDCH_DIMENSIONAL_SCHEMA is equal to PRO_CHM_ANG_X_D.
- PRO_E_RNDCH_COMPOUND_EXT_OPTIONS—Specifies the external options. This is an optional element and has the following elements:
  - PRO_E_RNDCH_AUTO_BLEND—Specifies the auto blend.
  - PRO_E_RNDCH_TERM_SURFACE—Specifies terminating surface.

- PRO_E_RNDCH_RADII—Specifies an array of radius PRO_E_RNDCH_RADIUS.
- PRO_E_RNDCH_AMBIGUITY—Specifies the ambiguity in the chamfer set.

> **Note**
>
> The chamfer set can contain ambiguity if the chamfer set contains chamfer pieces that co-exist and can be placed in various locations in the selected references and in part geometry.

Element Details of PRO_E_RNDCH_RADIUS for Chamfer

Each PRO_E_RNDCH_RADIUS has the following elements:

- PRO_E_RNDCH_LEG1—Specifies leg1 of the chamfer feature.
- PRO_E_RNDCH_LEG2—Specifies leg2 of the chamfer feature and is a required element if either PRO_E_RNDCH_DIMENSIONAL_SCHEMA is equal to PRO_CHM_D1_X_D2 or PRO_E_RNDCH_DIMENSIONAL_SCHEMA is equal to PRO_CHM_ANG_X_D.

Each PRO_E_RNDCH_LEG1 or PRO_E_RNDCH_LEG2 has the following elements:

- PRO_E_RNDCH_LEG_TYPE—Specifies leg type and is a mandatory element. It is of the following types:
  - PRO_ROUND_RADIUS_TYPE_VALUE
  - PRO_ROUND_RADIUS_THROUGH_POINT

The definition of ProRoundRadiusType is as follows:

- PRO_E_RNDCH_LEG_VALUE—Specifies leg value and is required if PRO_E_RNDCH_LEG_TYPE is equal to PRO_ROUND_RADIUS_TYPE_VALUE.
- PRO_E_RNDCH_REFERENCE_EDGE—Specifies reference edge and is required if PRO_E_RNDCH_LEG_TYPE is equal to PRO_ROUND_RADIUS_THROUGH_POINT.
- PRO_E_RNDCH_REFERENCE_POINT—Specifies reference point and is required if PRO_E_RNDCH_LEG_TYPE is equal to PRO_ROUND_RADIUS_THROUGH_POINT.

Element Details of PRO_E_RNDCH_TRANSITION for Chamfer

Each PRO_E_RNDCH_TRANSITION represents user-defined transitions for the entire chamfer feature and consists of the following elements:

- PRO_E_RNDCH_TRANS_TYPE—Specifies the type of the transition type. Valid values are:
  - PRO_ROUND_TRANS_INTERSECT—Used to extend two or more overlapping chamfer pieces towards each other until they merge forming a sharp boundary.
  - PRO_ROUND_TRANS_BLEND—Used to create a fillet surface between the chamfer pieces using an edge reference.
  - PRO_ROUND_TRANS_STOP—Used to terminate the chamfer geometry at the specified datum point or datum plane.
  - PRO_ROUND_TRANS_CONTINUE—Used to extend chamfer geometry into two chamfer pieces.
  - PRO_ROUND_TRANS_PATCH—Used to create a patched surface at the location where three or four chamfer pieces overlap.
  - PRO_ROUND_TRANS_BLEND_3SRF—Used to create a triangular patch as a transition of three chamfers.
  - PRO_ROUND_TRANS_PLANE_CORNER—Used to chamfer the corner transition formed by three overlapping chamfer pieces with a plane.
  - PRO_ROUND_TRANS_RBALL—Used to create a rolling ball transition for three or more chamfers.
  - PRO_ROUND_TRANS_STOP_0_SIDE—Used to terminate the chamfer using geometry configured by Creo Parametric.
  - PRO_ROUND_TRANS_STOP_1_SIDE—Used to terminate the chamfer using geometry configured by Creo Parametric.
  - PRO_ROUND_TRANS_STOP_2_SIDE—Used to terminate the chamfer using geometry configured by Creo Parametric.
  - PRO_ROUND_TRANS_STOP_AT_REF—Used to terminate the chamfer geometry at the selected datum point or datum plane.

Note

Only some of the transition types listed above are available for a given context.

- PRO_E_RNDCH_TRANS_CAP—Specifies the capping surface for chamfer pieces of the chamfer feature. It has the following values:
  - PRO_ROUND_CAPPING_SURF_DISABLE = PRO_B_FALSE.
  - PRO_ROUND_CAPPING_SURF_ENABLE = PRO_B_TRUE.
- PRO_E_RNDCH_TRANS_PATCH_DATA—Specifies the patch data and is required if PRO_E_RNDCH_TRANS_TYPE is equal to PRO_ROUND_TRANS_PATCH. It has the following elements:
  - PRO_E_RNDCH_TRANS_PATCH_REF_SURF—Specifies the arc surface, which indicates that a valid surface reference has been selected to place a fillet for the active patch transition. It should have the value as PRO_SURFACE only.
  - PRO_E_RNDCH_TRANS_PATCH_RAD_OPT—Specifies the arc radius options. It indicates the fillet radius for the active Patch transition and has following options:
    - PRO_ROUND_TRANS_RADIUS_ENTER_VALUE— Specifies a new radius value.
    - PRO_ROUND_TRANS_RADIUS_AUTOMATIC— Specifies the most recently used value.
    - PRO_E_RNDCH_TRANS_ARC_RADIUS—Specifies the arc radius.
- PRO_E_RNDCH_TRANS_STOP_DATA—Specifies the capping surface. It has the following elements:
  - PRO_E_RNDCH_TRANS_STOP_REF_TYPE—Specifies the reference type. Valid values are:
    - PRO_ROUND_TRANS_REF_NO_REF
    - PRO_ROUND_TRANS_REF_GEOM
    - PRO_ROUND_TRANS_REF_PNTVTX
    - PRO_ROUND_TRANS_REF_DTMPLN
    - PRO_ROUND_TRANS_REF_ISOLINE
    - PRO_E_RNDCH_TRANS_STOP_REFERENCE—Specifies the references for the active stop at reference transition. It can either be PRO_SURFACE or PRO_POINT.

Creating a Chamfer

Function Introduced:

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Chamfer based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Note

In Pro/ENGINEER Wildfire 2.0,

1. Pro/TOOLKIT does not support the temporary geometry required for user-specified ambiguity and non-default transitions. Therefore, these elements cannot be used for creation of new chamfers.
2. If transitions are specified in the input element tree, a chamfer feature with the default transition will be created.
3. In case of ambiguous situation (where more than one valid solutions exist, e.g. for surface-surface chamfer - surfaces having discontinuous edges of intersection), a chamfer feature with default solution will be created.

Redefining a Chamfer

Function Introduced:

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Chamfer based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Note

In Pro/ENGINEER Wildfire 2.0,

1. A chamfer feature having default transition, can not be redefined to have any transition.
2. A chamfer feature having a Pro/ENGINEER user interface defined transition can be redefined to other type of transition, for example, from intersect type to corner type. The input element tree must have a valid transition of the required type.

Accessing a Chamfer

Function Introduced:

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Chamfer and to retrieve the element tree description of a Chamfer. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Example 2: Sample code for creation of a Chamfer Feature

The sample code in `UgChamferTemplate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a Chamfer feature. It includes all possible element assignments. By following the instructions for the feature you want to create, it should be possible to remove element settings not appropriate for your use.

Example 3: Sample code for creation of a Edge Chamfer Feature

The sample code in `UgEdgeChamferCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a Edge Chamfer feature. The user is prompted to select an edge on which the chamfer will be created.

Edit Chamfer Feature

As a part of the flexible modeling capabilities the Edit Chamfer feature allows you to modify the existing chamfer geometry of an edge chamfer. You can modify the distance (D) from the edge along each surface, the D1 and D2 values, and so on and also the offset distance (O) from the edge along each surface. The ability to switch from chamfer by offset (O) to chamfer by extension (D) will have the same restrictions as in the chamfer feature.

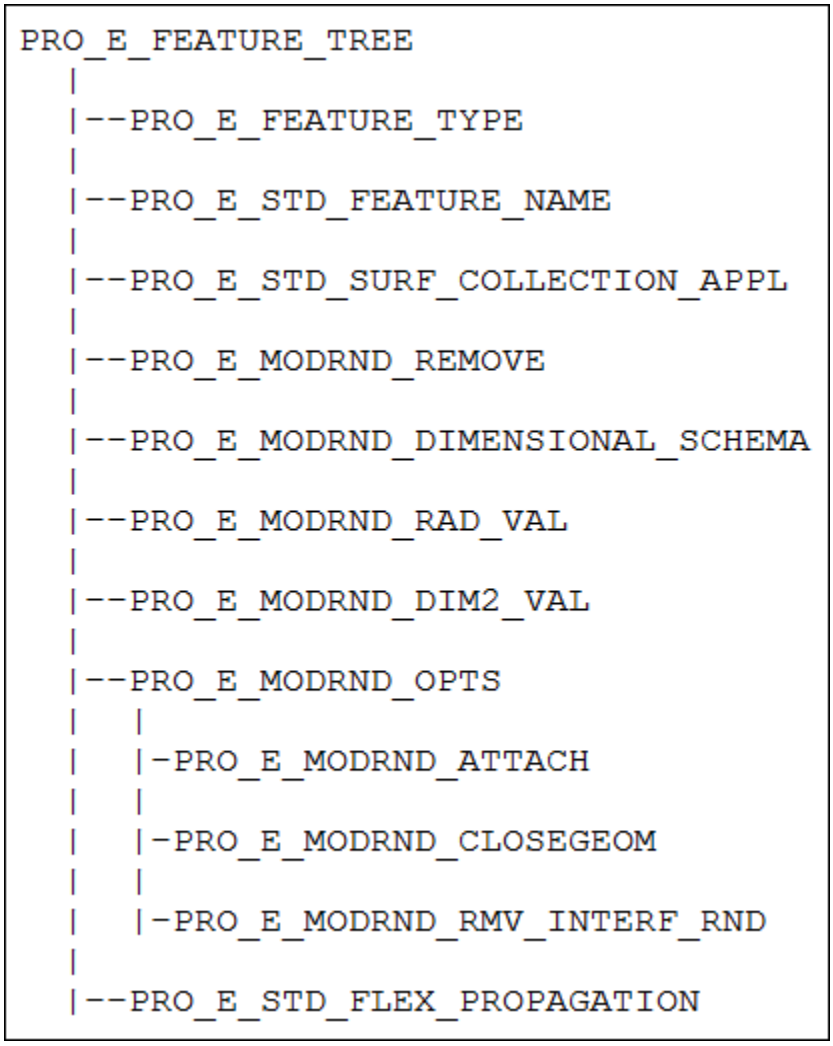
To create the Edit Chamfer feature follow the steps:

1. Identify the chamfer geometry to be modified and its D or O value.
2. Modify the specified chamfer geometry by using Creo Parametric. The application removes the identified chamfer geometry and recreates the chamfered edges with the desired values. The feature IDs of recreated chamfers are updated.

Feature Element Tree for Edit Chamfer Feature

The element tree for an Edit Chamfer feature is documented in the header file `ProModifyChamfer.h` and is as shown in the following figure.

Edit Chamfer Element Tree



The following table describes the elements in the element tree:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of feature. The value of this feature must be PRO_FEAT_MOD_CHAMFER.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Mandatory element. Specifies the name of the feature. The default value is EDIT_CHAMFER.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies a collection of reference surfaces that include the chamfer geometry to be modified.

Element ID	Data Type	Description
		In Creo Parametric TOOLKIT 7.0.0.0 and later, you can select the reference surfaces from both different solid bodies as well as quilts. The resulting geometry is attached back to the same solid body or quilt from where the referenced surfaces were selected.
PRO_E_MODRND_REMOVE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies if the existing chamfer must be removed.
PRO_E_MODRND_ DIMENSIONAL_SCHEMA	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of chamfer or the dimension schema. The values for this element are defined by the enumerated type ProChmSchema. For more information on the chamfer types, refer to section Element Details of PRO_E_RNDCH_SET for Chamfer on page 918.
PRO_E_MODRND_RAD_VAL	PRO_VALUE_TYPE_BOOLEAN	Mandatory element. Specifies the first distance or the offset value depending on the type of chamfer defined by the enumerated type ProChmSchema. Specify a value in the range of [(part epsilon / 10.0), 1.0e+06] for D type schemas and [-1.0e+06, 1.0e+06] for O type schemas.
PRO_E_MODRND_DIM2_VAL	PRO_VALUE_TYPE_BOOLEAN	Specifies the second distance, offset or angular value of the chamfer. Mandatory element if the value of the enumerated type ProChmSchema is one of the following: • PRO_CHM_D1_X_D2 • PRO_CHM_O1_X_O2 • PRO_CHM_ANG_X_D Specify a value in the range of:

Element ID	Data Type	Description
		• `[(part epsilon / 10.0), 1.0e+06]` for D type schemas • `[-1.0e+06, 1.0e+06]` for O type schemas • `[0, 180]` for chamfer angle ANG.
`PRO_E_MODRND_OPTS`	`Compound`	Mandatory element. Specifies the modification options for each chamfer to be recreated.
`PRO_E_MODRND_ATTACH`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies if the chamfer geometry must be attached to the selected reference surface once the chamfer is recreated. The values for this element are defined by the enumerated type `ProModRndAttach`.
`PRO_E_MODRND_CLOSEGEOM`	`PRO_VALUE_TYPE_INT`	Mandatory element. Use this element only if the chamfers are recreated as new surfaces or quilts, that is, if the element `PRO_E_MODRND_ATTACH` has the value `PRO_MODRND_DONOT_ATTACH_GEOM`. Specifies if end surfaces must be created for the recreated chamfer geometry. The values for this element are defined by the enumerated type `ProModRndCloseGeom`.

Element ID	Data Type	Description
PRO_E_MODRND_RMV_INTERF_RND	PRO_VALUE_TYPE_INT	Mandatory element. Specifies if the interfering chamfers must be removed and recreated in order to recreate the modified chamfer geometry. The values for this element are defined by the enumerated type ProModRndRecrRounds.
PRO_E_STD_FLEX_PROPAGATION	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies a Pattern feature, a Symmetry Recognition feature, or a Mirror Geometry feature that contains the reference surfaces specified by the element PRO_E_STD_SURF_COLLECTION_APPL. If such surfaces exist, then the modification of the chamfer is propagated to all corresponding surfaces in the instances of the specified feature to maintain the pattern or symmetry.

Corner Chamfer Feature

Feature Element Tree for Corner Chamfer

The element tree for a corner chamfer is documented in the header file ProCornerChamfer.h, and has a simple structure. The following figure demonstrates the feature element tree structure:

Corner Chamfer Element Tree

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_STD_CORNER_CHAMF_CORNER
   |
   |--PRO_E_STD_CORNER_CHAMF_EDGE1
   |
   |--PRO_E_STD_CORNER_CHAMF_EDGE2
   |
   |--PRO_E_STD_CORNER_CHAMF_EDGE3
```

The elements in this tree are described as follows:

- PRO_E_STD_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_CORN_CHAMF.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature. The default value for this element is CORNER_CHAMFER_#, where # specifies the feature number.
- PRO_E_STD_CORNER_CHAMF_CORNER—Specifies the vertex on which the corner chamfer is placed and can be selection of the type PRO_EDGE, PRO_EDGE_PNT, PRO_EDGE_START or PRO_EDGE_END. When using PRO_EDGE and PRO_EDGE_PNT type of reference, an appropriate parameter should be set using the function ProReferenceParamsSet().
- PRO_E_STD_CORNER_CHAMF_EDGE1— Specifies the first distance value from the vertex to the chamfer along first direction edge.
- PRO_E_STD_CORNER_CHAMF_EDGE2— Specifies the second distance value from the vertex to the chamfer along the second direction edge.
- PRO_E_STD_CORNER_CHAMF_EDGE3— Specifies the third distance value from the vertex to the chamfer along the third direction edge.

Creating a Corner Chamfer

Function Introduced:

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a Corner Chamfer based on element tree input. For more information about ProFeatureCreate(), refer to the section Overview of Feature Creation on page 765 in the Element Trees: Principles of Feature Creation on page 764.chapter.

Redefining a Corner Chamfer

Function Introduced:

- **ProFeatureRedefine()**

Use the function ProFeatureRedefine() to redefine a Corner Chamfer based on the changes made in the element tree. For more information about ProFeatureRedefine(), refer to the section Feature Redefine on page 786 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Corner Chamfer

Function Introduced:

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Corner Chamfer and to retrieve the element tree description of a Corner Chamfer. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 in the Element Trees: Principles of Feature Creation on page 764 chapter.

40

Element Trees: Hole

This chapter describes the programmatic creation of Hole features using the Creo Parametric TOOLKIT include file `ProHole.h`.

We recommend you read the section, Overview of Feature Creation on page 765 in the chapter, Element Trees: Principles of Feature Creation on page 764 It provides the necessary background for creating features using Creo Parametric TOOLKIT.

Overview

Creo Parametric TOOLKIT supports four types of Holes:

- Straight
- Standard
- Sketched
- Custom

The Standard Hole type is sub-divided into two categories:

- Standard Clearance Hole
- Standard Threaded Hole

This chapter details the procedure and the sequence of the creation of the element tree for all of the above types.

All Hole types and placement types require entry of specific elements during element tree creation. Elements must be entered in the specified order.

To create a Hole feature, first add to the element tree all elements related to the hole type. Then, add the elements required for Hole placement. Creating Sketched Holes uses techniques similar to creation of the other sketched features (see Element Trees: Sketched Features on page 1004).

Note

All angle elements are specified in degrees.

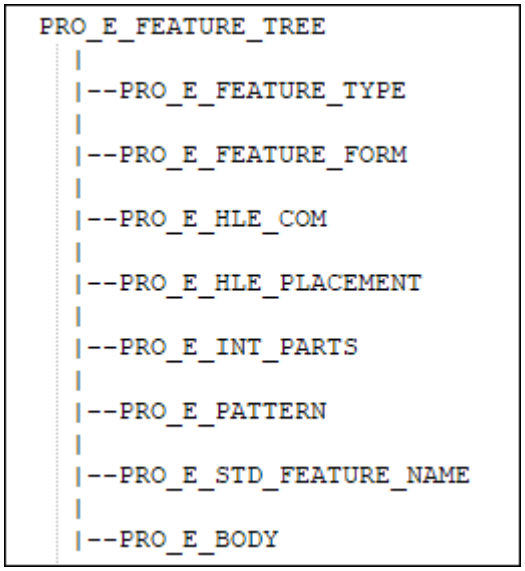
You can use Intent Datums such as Intent Point, Intent Axis, and Intent Plane for hole placement in parts.

Lightweight holes can be created only in parts and not in assemblies. You can toggle between a lightweight and regular hole, only in case of simple holes and not in sketched, standard, or custom holes.

Feature Element Tree for Hole Features

The element tree for the Hole feature is documented in the Creo Parametric TOOLKIT header file `ProHole.h`.

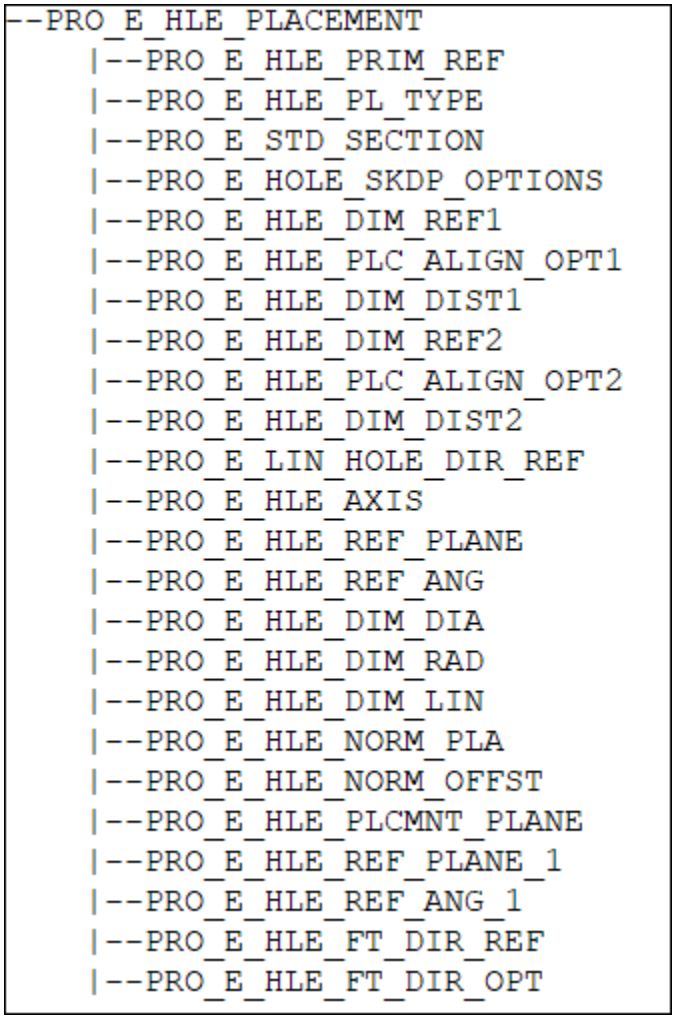
Feature Element Tree for Hole Feature



Common Elements for Hole Types

```
--PRO_E_HLE_COM
    |--PRO_E_HLE_TYPE_NEW
    |--PRO_E_HLE_STAN_TYPE
    |--PRO_E_HLE_THRDSERIS
    |--PRO_E_HLE_FITTYPE
    |--PRO_E_HLE_SCREWSIZE
    |--PRO_E_HLE_ADD_THREAD
    |--PRO_E_HLE_ADD_CBORE
    |--PRO_E_HLE_ADD_CSINK
    |--PRO_E_HLE_MAKE_LIGHTWT
    |
    |--PRO_E_DIAMETER
    |
    |--PRO_E_HOLE_STD_DEPTH
    |     |--PRO_E_HOLE_DEPTH_TO
    |     |     |--PRO_E_HOLE_DEPTH_TO_TYPE
    |     |     |--PRO_E_EXT_DEPTH_TO_VALUE
    |     |     |--PRO_E_EXT_DEPTH_TO_REF
    |     |--PRO_E_HOLE_DEPTH_FROM
    |           |--PRO_E_HOLE_DEPTH_FROM_TYPE
    |           |--PRO_E_EXT_DEPTH_FROM_VALUE
    |           |--PRO_E_EXT_DEPTH_FROM_REF
    |
    |--PRO_E_HLE_HOLEDIAM
    |--PRO_E_HLE_DEPTH
    |--PRO_E_HLE_CSINKANGLE
    |--PRO_E_HLE_CBOREDEPTH
    |--PRO_E_HLE_CBOREDIAM
    |--PRO_E_HLE_CSINKDIAM
    |--PRO_E_HLE_DEPTH_DIM_TYPE
    |--PRO_E_HLE_THRD_DEPTH
    |--PRO_E_HLE_THRDDEPTH
    |--PRO_E_HLE_DRILLANGLE
    |--PRO_E_HLE_DRILLDEPTH
    |--PRO_E_HLE_TAPERED_STRT_DEPTH_OPT
    |--PRO_E_STD_HOLE_DEPTH_REF
    |--PRO_E_HLE_ADD_TAPERED_TIP_ANGLE
    |--PRO_E_HLE_TAPERED_STRT_DIA
    |--PRO_E_HLE_TAPERED_STRT_DEPTH
    |--PRO_E_HLE_TAPERED_TIP_ANGLE
    |--PRO_E_SKETCHER
    |--PRO_E_HLE_CRDIR_FLIP
    |--PRO_E_HLE_ADD_EXIT_CSINK
    |--PRO_E_HLE_EXIT_CSINKANGLE
```

Common Elements for Hole Placement



Feature Element Data Types

The following table lists data types for hole type and placement elements.

Element values must be of the specified type.

Hole Element Table

Element Id	Element Name	Data Type
PRO_E_FEATURE_TYPE	Feature Type	PRO_VALUE_TYPE_INT
PRO_E_FEATURE_FORM	Feature Form	PRO_VALUE_TYPE_INT
PRO_E_HLE_COM	Hole	Compound
PRO_E_HLE_TYPE_NEW	Hole Type	PRO_VALUE_TYPE_INT
PRO_E_HLE_STAN_TYPE	Standard Type	PRO_VALUE_TYPE_INT
PRO_E_HLE_THRDSERIS	Thread Series	PRO_VALUE_TYPE_INT
PRO_E_HLE_FITTYPE	Fit Type	PRO_VALUE_TYPE_INT

Element Id	Element Name	Data Type
PRO_E_HLE_SCREWSIZE	Screw Size	PRO_VALUE_TYPE_INT
PRO_E_HLE_ADD_THREAD	Add Thread	PRO_VALUE_TYPE_INT
PRO_E_HLE_ADD_CBORE	Add Counterbore	PRO_VALUE_TYPE_INT
PRO_E_HLE_ADD_CSINK	Add Countersink	PRO_VALUE_TYPE_INT
PRO_E_HLE_MAKE_LIGHTWT	Make lightweight hole	PRO_VALUE_TYPE_INT (It is given by the enumerated type ProHleLightWtFlag)
PRO_E_DIAMETER	Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HOLE_STD_DEPTH	Depth Element	Compound
PRO_E_HOLE_DEPTH_TO	Depth Two	Compound
PRO_E_HOLE_DEPTH_TO_TYPE	Depth Two	PRO_VALUE_TYPE_INT
PRO_E_EXT_DEPTH_TO_VALUE	Depth Value	PRO_VALUE_TYPE_DOUBLE
PRO_E_EXT_DEPTH_TO_REF	Reference	PRO_VALUE_TYPE_SELECTION
PRO_E_HOLE_DEPTH_FROM	Depth One	Compound
PRO_E_HOLE_DEPTH_FROM_TYPE	Depth One	PRO_VALUE_TYPE_INT
PRO_E_EXT_DEPTH_FROM_VALUE	Depth Value	PRO_VALUE_TYPE_DOUBLE
PRO_E_EXT_DEPTH_FROM_REF	Reference	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_HOLEDIAM	Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DEPTH	Depth	PRO_VALUE_TYPE_INT
PRO_E_HLE_CSINKANGLE	Csink Angle	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_CBOREDEPTH	Counterbore Depth	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_CBOREDIAM	Counterbore Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_CSINKDIAM	Csink Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DEPTH_DIM_TYPE	Depth Dim Scheme	PRO_VALUE_TYPE_INT
PRO_E_HLE_THRD_DEPTH	Thread Depth	PRO_VALUE_TYPE_INT
PRO_E_HLE_THRDDEPTH	Thread Depth	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DRILLANGLE	Drillhead Angle	PRO_VALUE_TYPE_DOUBLE
RO_E_HLE_DRILLDEPTH	Drill Depth	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_TAPERED_STRT_DEPTH_OPT	Straight Depth Options	PRO_VALUE_TYPE_INT
PRO_E_STD_HOLE_DEPTH_REF	Reference	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_ADD_TAPERED_TIP_ANGLE	Tapered Tip	PRO_VALUE_TYPE_INT
PRO_E_HLE_TAPERED_STRT_DIA	Straight Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_TAPERED_STRT_DEPTH	Straight Depth	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_TAPERED_TIP_ANGLE	Tapered Tip Angle	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_SKETCHER	Sketcher	N/A

Element Id	Element Name	Data Type
PRO_E_HLE_CRDIR_FLIP	Creation Direction	PRO_VALUE_TYPE_INT
PRO_E_HLE_ADD_EXIT_CSINK	Add Exit Csink	PRO_VALUE_TYPE_INT
PRO_E_HLE_EXIT_CSINKANGLE	Exit Csink Angle	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_EXIT_CSINKDIAM	Exit Csink Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_ADD_NOTE	Add Hole Note	PRO_VALUE_TYPE_INT
PRO_E_HOLE_NOTE	Hole Note	The element is not accessible through Creo Parametric TOOLKIT
PRO_E_HLE_TOP_CLEARANCE	Top Clearance	PRO_VALUE_TYPE_INT
PRO_E_HLE_THRDTOSEL	Reference	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_PLACEMENT	Placement	N/A
PRO_E_HLE_PRIM_REF	Primary Reference	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_PL_TYPE	Placement Options	PRO_VALUE_TYPE_INT
PRO_E_HLE_DIM_REF1	DimensionRef 1	PRO_VALUE_TYPE_SELECTION
PRO_E_STD_SECTION	Section	Compound
PRO_E_HOLE_SKDP_OPTIONS	Use Options	PRO_VALUE_TYPE_INT. This element allows you to select sketched datum point options on which the holes can be placed. The element PRO_E_HOLE_SKDP_OPTIONS is defined by the enumerated data type ProHleSkdpOption and the valid values are: • PRO_HLE_SKDP_POINT_OPT—Points entities from sketch • PRO_HLE_SKDP_ENDPT_OPT—End points of line entities from sketch • PRO_HLE_SKDP_MID_OPT—Mid Points of line entities from sketch
PRO_E_HLE_PLC_ALIGN_OPT1	Alignment for placement reference 1	PRO_VALUE_TYPE_INT
PRO_E_HLE_DIM_DIST1	Distance 1	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DIM_REF2	DimensionRef 2	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_PLC_ALIGN_OPT2	Alignment for placement reference 2	PRO_VALUE_TYPE_INT
PRO_E_LIN_HOLE_DIR_REF	Reference Direction	PRO_VALUE_TYPE_SELECT
PRO_E_HLE_DIM_DIST2	Distance 2	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_AXIS	Axis	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_REF_PLANE	Reference Plane	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_REF_ANG	Angle	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DIM_DIA	Diameter	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DIM_RAD	Radius	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_DIM_LIN	Linear Distance	PRO_VALUE_TYPE_DOUBLE

Element Id	Element Name	Data Type
PRO_E_HLE_NORM_PLA	Normal Plane	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_NORM_OFFST	Offset	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_PLCMNT_PLANE	Placement Plane	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_REF_PLANE_1	Reference Plane	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_REF_ANG_1	Angle	PRO_VALUE_TYPE_DOUBLE
PRO_E_HLE_FT_DIR_REF	Direction reference	PRO_VALUE_TYPE_SELECTION
PRO_E_HLE_FT_DIR_OPT	Direction option	PRO_VALUE_TYPE_INT
PRO_E_INT_PARTS	Intsct Parts	N/A
PRO_E_PATTERN	Pattern	N/A
PRO_E_STD_FEATURE_NAME	Feature Name	PRO_VALUE_TYPE_WSTRING
PRO_E_BODY	Compound	Compound element that holds Body options. For more information, refer to the ProBodyOpts.h element tree.
PRO_E_BODY_USE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the bodies on which the hole feature is created. The valid values are: • PRO_BODY_USE_ALL—Hole is creating on all the existing bodies. **Note** This option is not available for the following hole depth options: ○ To Next ○ Through Until • PRO_BODY_USE_SELECTED—Hole is created on the selected bodies.
PRO_E_BODY_SELECT	PRO_VALUE_TYPE_SELECTION	Specifies the reference to the selected body. Mandatory if the value of PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED. **Note** Multiple references are allowed.

Common Element Values

All holes require definition of the feature type and feature form. The following table shows valid values for the common elements in the hole element tree.

Common Element Values

Element ID	Value
PRO_E_FEATURE_TYPE	PRO_FEAT_HOLE
PRO_E_FEATURE_FORM	PRO_HLE_TYPE_STRAIGHT (for straight holes) PRO_HLE_TYPE_SKETCHED (for other hole types)
PRO_E_STD_FEATURE_NAME	Wstring (feature name)
PRO_E_BODY	Compound element that holds Body options.

PRO_E_HLE_COM Values

Values required for PRO_E_HLE_COM compound element vary for different hole types. The following tables show the PRO_E_HLE_COM element values required to define different hole types. Be sure to enter the elements into the element tree in the order specified by these tables.

Straight Hole

The following table shows elements for creating a straight hole.

Straight Hole Elements

Element	Status
PRO_E_HLE_TYPE_NEW	PRO_HLE_NEW_TYPE_STRAIGHT
PRO_E_HLE_MAKE_LIGHTWT	Mandatory
PRO_E_DIAMETER	Mandatory
PRO_E_HOLE_STD_DEPTH	Mandatory
PRO_E_HOLE_DEPTH_TO	Mandatory
PRO_E_HOLE_DEPTH_TO_TYPE	Mandatory
PRO_E_EXT_DEPTH_TO_VALUE	Depends on PRO_E_HOLE_DEPTH_TO_TYPE
PRO_E_EXT_DEPTH_TO_REF	Depends on PRO_E_HOLE_DEPTH_TO_TYPE
PRO_E_HOLE_DEPTH_FROM	Mandatory
PRO_E_HOLE_DEPTH_FROM_TYPE	Mandatory
PRO_E_EXT_DEPTH_FROM_VALUE	Depends on PRO_E_HOLE_DEPTH_FROM_TYPE
PRO_E_EXT_DEPTH_FROM_REF	Depends on PRO_E_HOLE_DEPTH_FROM_TYPE
PRO_E_HLE_TOP_CLEARANCE	Mandatory

Figure Straight Hole with Linear Placement on page 940 shows code for creating a straight hole with linear placement and through-all depth. The hole has 100 units diameter, and is placed 100 units distant from the first reference and 200 units distance from the second.

The function ProDemoHoleCreate() builds the complete element tree serially. First add all elements required for the straight hole under the PRO_E_HLE_COM element. Then enter the placement elements under the PRO_E_HLE_PLACEMENT element. Use element PRO_E_HOLE_DEPTH_TO_TYPE to specify the hole as 'through all'.

The function UserElemtreeElementAdd() is a small utility that add an element to the element tree.

Straight Hole with Linear Placement

The sample code in UgHoleCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a straight linear hole.

Sketched Hole

Creo Parametric TOOLKIT supports two methods for creating sketched holes. The first is as described in Element Trees: Sketched Features on page 1004, the second uses the function `ProFeatureCreate()` more directly.

The following table describes the required elements for sketched hole features.

Sketched Hole Elements

Element	Status
`PRO_E_HLE_TYPE_NEW`	`PRO_HLE_NEW_TYPE_SKETCH`
`PRO_E_SKETCHER`	Mandatory
`PRO_E_HLE_CRDIR_FLIP`	Mandatory
`PRO_E_HLE_TOP_CLEARANCE`	Mandatory

Procedure Using Techniques from Creating Sketched Features

1. Add the required elements for the sketched feature as outlined in the table in section Sketched Hole on page 941.
2. Add all the placement elements.
3. Set the argument `ProFeatureCreateOptions` for `ProFeatureCreate()` to `PRO_FEAT_CR_INCOMPLETE_FEAT` and call `ProFeatureCreate()` with the created element tree.
4. Fetch the section handle for the section of the incomplete feature, using the sequence of calls `ProElempathAlloc()`, `ProElempathDataSet()`, and `ProFeatureElemValueGet()`.
5. Create a 2D revolved section with the retrieved section handle. Add the center-line for the axis of revolution as required for the section for revolved feature in Creo Parametric user interface.
6. Attach the new section to the element tree, then call `ProFeatureRedefine()` with the element tree created in these steps.

Refer to Sketched Hole with Conventional Approach on page 941 for a code example of this technique of hole creation.

Sketched Hole with Conventional Approach

Example 1: Creating a Standard Sketched Hole with Linear Placement

The sample code in `UgHoleCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a standard sketched hole with linear placement. In the conventional approach:

- Use `ProFeatureCreate()` to create incomplete feature
- Use the feature handle to get the section handle
- Build the section
- Give a call to `ProFeatureRedefine()` to redefine
- Complete the feature.

Procedure Using ProFeatureCreate()

In this approach to sketched hole creation, populate the required elements in the element tree (as shown in the table in section Sketched Hole on page 941), and then call `ProFeatureCreate()`.

Refer to Standard Threaded Hole on page 942 for a code example for creation of this type of hole.

Refer to Sketched Hole with ProFeatureCreate() on page 942 for a code example of this technique of hole creation.

Sketched Hole with ProFeatureCreate()

Example: Creating a Standard Sketched Hole with Linear Placement

The sample code in `UgHoleCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a standard sketched hole with linear placement. Using new approach:

- Create the complete element tree with the sketcher element
- Call `ProFeatureCreate()` to create the hole feature

Standard Threaded Hole

The following table shows elements for creating a standard threaded hole.

Refer to Example 2: Creating a Standard Threaded Hole with Linear Placement on page 943 for a code example on creating this type of hole.

Element	Status
`PRO_E_HLE_TYPE_NEW`	Mandatory
`PRO_E_HLE_STAN_TYPE`	Mandatory
`PRO_E_HLE_THRDSERIS`	Mandatory
`PRO_E_HLE_FITTYPE`	Mandatory: set to `PRO_HLE_CLOSE_FIT`
`PRO_E_HLE_SCREWSIZE`	Mandatory
`PRO_E_HLE_ADD_THREAD`	Mandatory
`PRO_E_HLE_ADD_CBORE`	Mandatory
`PRO_E_HLE_ADD_CSINK`	Mandatory

Element	Status
PRO_E_HLE_HOLEDIAM	Mandatory
PRO_E_HLE_DRILLANGLE	Required for variable depth hole
PRO_E_HLE_CSINKANGLE	Required for countersink option
PRO_E_HLE_CBOREDEPTH	Required for counterbore option
PRO_E_HLE_CBOREDIAM	Required for counterbore option
PRO_E_HLE_CSINKDIAM	Required for countersink option
PRO_E_HLE_THRDDEPTH	Mandatory, even for a non-threaded hole or a thru-threaded hole. This element is required. If not added, hole creation will succeed but the feature cannot be redefined in the Creo Parametric user interface.
PRO_E_HLE_DRILLDEPTH	Mandatory, even for a through-all hole. This element is required. If not added, hole creation will succeed but the feature cannot be redefined in the Creo Parametric user interface.
PRO_E_HLE_THRD_DEPTH	Mandatory
PRO_E_HLE_TAPERED_STRT_DEPTH_OPT	Mandatory
PRO_E_HLE_DEPTH	Mandatory
PRO_E_STD_HOLE_DEPTH_REF	Depends on PRO_E_HLE_DEPTH
PRO_E_HLE_ADD_TAPERED_TIP_ANGLE	Mandatory
PRO_E_HLE_TAPERED_STRT_DIA	Depends on PRO_E_HLE_TAPERED_STRT_DEPTH_OPT
PRO_E_HLE_TAPERED_STRT_DEPTH	Depends on PRO_E_HLE_TAPERED_STRT_DEPTH_OPT
PRO_E_HLE_TAPERED_TIP_ANGLE	Depends on PRO_E_HLE_ADD_TAPERED_TIP_ANGLE
PRO_E_HLE_DEPTH_DIM_TYPE	Depends on PRO_E_HLE_DEPTH
PRO_E_HLE_CRDIR_FLIP	Mandatory
PRO_E_HLE_ADD_EXIT_CSINK	Required for Thru all hole
PRO_E_HLE_EXIT_CSINKANGLE	Required for exit countersink option
PRO_E_HLE_EXIT_CSINKDIAM	Required for exit countersink option
PRO_E_HLE_ADD_NOTE	Required for Hole note
PRO_E_HOLE_NOTE	Depends on PRO_E_HLE_ADD_NOTE
PRO_E_HLE_TOP_CLEARANCE	Mandatory

Example 2: Creating a Standard Threaded Hole with Linear Placement

The sample code in UgHoleCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a standard threaded hole with linear placement.

Standard Clearance Hole

The following table outlines elements for standard clearance holes.

Standard Clearance Hole Elements

Element	Status
PRO_E_HLE_TYPE_NEW	Mandatory
PRO_E_HLE_STAN_TYPE	Mandatory
PRO_E_HLE_THRDSERIS	Mandatory
PRO_E_HLE_FITTYPE	Mandatory
PRO_E_HLE_SCREWSIZE	Mandatory
PRO_E_HLE_ADD_THREAD	Mandatory
PRO_E_HLE_ADD_CBORE	Mandatory
PRO_E_HLE_ADD_CSINK	Mandatory
PRO_E_HLE_HOLEDIAM	Mandatory
PRO_E_HLE_DRILLANGLE	Required for variable depth hole
PRO_E_HLE_CSINKANGLE	Required for countersink option
PRO_E_HLE_CBOREDEPTH	Required for counterbore option
PRO_E_HLE_CBOREDIAM	Required for counterbore option
PRO_E_HLE_CSINKDIAM	Required for countersink option
PRO_E_HLE_DEPTH	Mandatory
PRO_E_HLE_CRDIR_FLIP	Mandatory
PRO_E_HLE_ADD_EXIT_CSINK	Required for Thru all hole
PRO_E_HLE_EXIT_CSINKANGLE	Required for exit countersink option
PRO_E_HLE_EXIT_CSINKDIAM	Required for exit countersink option
PRO_E_HLE_ADD_NOTE	Required for Hole note
PRO_E_HOLE_NOTE	Depends on PRO_E_HLE_ADD_NOTE
PRO_E_HLE_TOP_CLEARANCE	Mandatory

Custom Hole

The following table outlines the elements for creating a custom hole.

Custom Hole Elements

Element	Status
PRO_E_HLE_TYPE_NEW	Mandatory
PRO_E_HLE_ADD_CBORE	Mandatory
PRO_E_HLE_ADD_CSINK	Mandatory
PRO_E_HLE_HOLEDIAM	Mandatory
PRO_E_HLE_DRILLANGLE	Required for variable depth hole
PRO_E_HLE_CSINKANGLE	Required for countersink option
PRO_E_HLE_CBOREDEPTH	Required for counterbore option
PRO_E_HLE_CBOREDIAM	Required for counterbore option
PRO_E_HLE_CSINKDIAM	Required for countersink option

Element	Status
PRO_E_HLE_DEPTH	Mandatory
PRO_E_HLE_DEPTH_DIM_TYPE	Depends on PRO_E_HLE_DEPTH
PRO_E_HLE_CRDIR_FLIP	Mandatory
PRO_E_HLE_ADD_EXIT_CSINK	Required for Thru all hole
PRO_E_HLE_EXIT_CSINKANGLE	Required for exit countersink option
PRO_E_HLE_EXIT_CSINKDIAM	Required for exit countersink option
PRO_E_HLE_TOP_CLEARANCE	Mandatory

Standard Tapered Hole

The following table outlines the elements for creating a standard tapered hole.

Standard Tapered Hole Elements

Element	Status
PRO_E_HLE_TYPE_NEW	Mandatory
PRO_E_HLE_STAN_TYPE	Mandatory
PRO_E_HLE_THRDSERIS	Mandatory
PRO_E_HLE_THRDSERIS	Mandatory
PRO_E_HLE_ADD_THREAD	Mandatory
PRO_E_HLE_ADD_CBORE	Mandatory
PRO_E_HLE_ADD_CSINK	Mandatory
PRO_E_HLE_HOLEDIAM	Mandatory
PRO_E_HLE_DRILLANGLE	Required for variable depth hole
PRO_E_HLE_CSINKANGLE	Required for countersink option
PRO_E_HLE_CBOREDEPTH	Required for counterbore option
PRO_E_HLE_CBOREDIAM	Required for counterbore option
PRO_E_HLE_CSINKDIAM	Required for countersink option
PRO_E_HLE_THRDDEPTH	Mandatory
PRO_E_HLE_DRILLDEPTH	Mandatory
PRO_E_HLE_TAPERED_STRT_DEPTH_OPT	It is an option for different straight drill depth types of type ProHleTaperStrDepType. The valid values are: • PRO_HOLE_NONE_DEPTH_TYPE • PRO_HOLE_BLIND_DEPTH_TYPE—blind • PRO_HOLE_THRUNEXT_DEPTH_TYPE—through next

Element	Status
	• PRO_HOLE_THRUALL_DEPTH_TYPE—through all • PRO_HOLE_THRUNTIL_DEPTH_TYPE—through until • PRO_HOLE_TOREF_DEPTH_TYPE—upto reference
PRO_E_HLE_ADD_TAPERED_TIP_ANGLE	It is an option for tapered tip of type ProHleAddTaperedTipAngFlag. The valid values are: • PRO_HLE_NO_TAPERED_TIP_ANGL—No tip angle in the tapered hole. • PRO_HLE_ADD_TAPERED_TIP_ANGLE—Add tip angle in the tapered hole.
PRO_E_HLE_TAPERED_STRT_DIA	Stores taper straight hole diameter double value.
PRO_E_HLE_TAPERED_STRT_DEPTH	Stores taper straight hole depth double value. Available for tapered hole, with blind depth option, that is • PRO_E_HLE_STAN_TYPE ==PRO_HLE_TAPERED_TYPE • PRO_E_HLE_TAPERED_STRT_DEPTH_OPT==HOLE_BLIND_DEPTH_TYPE
PRO_E_HLE_TAPERED_TIP_ANGLE	Stores tapered tip angle double value. Available for tapered tip option, that is • PRO_E_HLE_ADD_TAPERED_TIP_ANGLE==PRO_HLE_ADD_TAPERED_TIP_ANGLE
PRO_E_HLE_CRDIR_FLIP	Mandatory
PRO_E_HLE_ADD_EXIT_CSINK	Required for Thru all hole
PRO_E_HLE_EXIT_CSINKANGLE	Required for exit countersink option
PRO_E_HLE_EXIT_CSINKDIAM	Required for exit countersink option
PRO_E_HLE_ADD_NOTE	Required for Hole note
PRO_E_HOLE_NOTE	Depends on PRO_E_HLE_ADD_NOTE

Element	Status
PRO_E_HLE_TOP_CLEARANCE	Mandatory
PRO_E_HLE_THRDTOSEL	Stores reference when PRO_E_HLE_THRD_DEPTH==PRO_HLE_TO_SELECTED_THREAD

Valid PRO_E_HLE_COM Sub-Elements

The following table gives the description of all the elements for all the hole types.

Element ID	Comment/Description
Straight Hole	
PRO_E_HLE_TYPE_NEW = PRO_HLE_NEW_TYPE_STRAIGHT	
PRO_E_DIAMETER	Stores the diameter double value
PRO_E_HOLE_STD_DEPTH	Depth (compound element)
PRO_E_HOLE_DEPTH_TO	First Side depth info (compound element)
PRO_E_HOLE_DEPTH_TO_TYPE	Type ProHleStraightDepType/* Blind*/ PRO_HLE_STRGHT_BLIND_DEPTH /*Thru Next*/ PRO_HLE_STRGHT_THRU_NEXT_DEPTH /* Thru All*/ PRO_HLE_STRGHT_THRU_ALL_DEPTH /*Thru Until*/ PRO_HLE_STRGHT_THRU_UNTIL_DEPTH /*Upto Ref*/ PRO_HLE_STRGHT_UPTO_REF_DEPTH /*None */ PRO_HLE_STRGHT_NONE_DEPTH /*Symmetric*/ PRO_HLE_STRGHT_SYM_DEPTH
PRO_E_EXT_DEPTH_TO_VALUE	Stores variable depth double value when PRO_E_HOLE_DEPTH_TO_TYPE equals PRO_HLE_STRGHT_BLIND_DEPTH.
PRO_E_EXT_DEPTH_TO_REF	Stores the upto reference when PRO_E_HOLE_DEPTH_TO_TYPE is not PRO_HLE_STRGHT_BLIND_DEPTH and not PRO_HLE_STRGHT_NONE_DEPTH.
PRO_E_HOLE_DEPTH_FROM	Second Side depth info (compound element).
PRO_E_HOLE_DEPTH_FROM_TYPE	Type ProHleStraightDep /* Blind*/ PRO_HLE_STRGHT_BLIND_DEPTH /*Thru Next*/ PRO_HLE_STRGHT_THRU_NEXT_DEPTH /* Thru All*/ PRO_HLE_STRGHT_THRU_ALL_DEPTH /*Thru Until*/ PRO_HLE_STRGHT_THRU_UNTIL_DEPTH /*Upto Ref*/ PRO_HLE_STRGHT_UPTO_REF_DEPTH /*None */ PRO_HLE_STRGHT_NONE_DEPTH

Element ID	Comment/Description
	/*Symmetric*/ PRO_HLE_STRGHT_SYM_DEPT
PRO_E_EXT_DEPTH_FROM_VALUE	Stores variable depth double value when PRO_E_HOLE_DEPTH_FROM_TYPE equals PRO_HLE_STRGHT_BLIND_DEPTH.
PRO_E_EXT_DEPTH_FROM_REF	Stores the upto reference when PRO_E_HOLE_DEPTH_FROM_TYPE is not PRO_HLE_STRGHT_BLIND_DEPTH and is not PRO_HLE_STRGHT_NONE_DEPTH and not PRO_HLE_STRGHT_SYM_DEPTH.
Sketch Hole	
PRO_E_HLE_TYPE_NEW set to PRO_HLE_NEW_TYPE_SKETCH	
PRO_E_HLE_SKETCHER	2D Sketcher Element
PRO_E_HLE_CRDIR_FLIP	Direction of creation, type ProHleCrDir
Standard Hole	
PRO_E_HLE_TYPE_NEW set to PRO_HLE_NEW_TYPE_STANDARD PRO_E_STAN_TYPE = PRO_HLE_TAPPED_TYPE /* Tapped hole */ = PRO_HLE_CLEARANCE_TYPE/* Clearance hole */	
PRO_E_HLE_STAN_TYPE	type ProHleStandType
PRO_E_HLE_THRDSERIS	Integer. The *.hol files get loaded as specified in Hole Parameter Files on page 955. From the *.holfiles, different THREAD_SERIES information are gathered and a list is formed. This element stores the current index to the list.
PRO_E_HLE_FITTYPE	type ProHleFittype. Available for clearance hole (when PRO_E_HLE_STAN_TYPE is PRO_HLE_CLEARANCE_TYPE). /* Close Fit */ PRO_HLE_CLOSE_FIT /* Free Fit */ PRO_HLE_FREE_FIT /* Medium Fit */ PRO_HLE_MEDIUM_FIT
PRO_E_HLE_SCREWSIZE	Integer Stores an index to the screw_size list. Selecting a thread series, choose one of the .hol files. From that file screw-size list is extracted.
PRO_E_HLE_DEPTH	It is an option for different type drill depth, that is, of type ProHleStdDepType. **Note** PRO_HLE_STD_VAR_DEPTH is not available for clearance hole (not for PRO_E_HLE_STAN_TYPE == PRO_HLE_CLEARANCE_TYPE).
PRO_E_STD_HOLE_DEPTH_REF	Stores reference, when PRO_E_HLE_DEPTH equals toPRO_HLE_STD_THRU_UNTIL_DEPTH

Element ID	Comment/Description
	or PRO_HLE_STD_TO_SEL_DEPTH
PRO_E_HLE_HOLEDIAM	Stores Drill Diameter double value. See Hole Diameter.
PRO_E_HLE_DRILLANGLE	Stores Drill Angle. Double value. Available for tapped hole with variable depth (when PRO_E_HLE_STAN_TYPE equals PRO_HLE_TAPPED_TYPE and PRO_E_HLE_DEPTH is PRO_HLE_STD_VAR_DEPTH.
PRO_E_HLE_ADD_THREAD	Option for adding thread. Available for tapped hole (when PRO_E_HLE_STAN_TYPEequals PRO_HLE_TAPPED_TYPE). Type ProHleAddThrdFlag. For add thread option it's value is PRO_HLE_ADD_THREAD. For no thread option, the value is PRO_HLE_NO_THREAD.
PRO_E_HLE_THRD_DEPTH	Option for different type of thread depth. Type ProHleThrdDepType. Available for tapped hole with thread option, when PRO_E_HLE_STAN_TYPE equals PRO_HLE_TAPPED_TYPE and PRO_E_HLE_ADD_THREAD equals PRO_HLE_ADD_THREAD. **Note** All options are available in both assembly & part level.
PRO_E_HLE_THRDDEPTH	Stores thread depth. Double value. Available for tapped hole, with variable thread option. That is, when PRO_E_HLE_STAN_TYPE equals PRO_HLE_TAPPED_TYPE, PRO_E_HLE_ADD_THREAD equals PRO_HLE_ADD_THREAD, and PRO_E_HLE_THRD_DEPTHequals PRO_HLE_VARIABLE_THREAD.
PRO_E_HLE_ADD_CBORE	Option for Counter Bore. Type ProHleAddCboreFlag. For counter bore it's value is PRO_HLE_ADD_CBORE. For the no counterbore option, set to PRO_HLE_NO_CBORE.
PRO_E_HLE_CBOREDEPTH	Stores counterbore depth. Double value. Available for counterbore option, when PRO_E_HLE_ADD_CBORE equals PRO_HLE_ADD_CBORE.
PRO_E_HLE_CBOREDIAM	Stores counterbore diameter. Double value. Available for counterbore option, when PRO_E_HLE_ADD_CBORE is PRO_HLE_ADD_CBORE.
PRO_E_HLE_ADD_CSINK	It is an option for Counter Sink. Type ProHleAddCsinkFlag. For counter sink it's value is PRO_HLE_ADD_CSINK. For no countersink, set to PRO_HLE_NO_CSINK.
PRO_E_HLE_CSINKANGLE	Stores counter sink angle. Double value. Available for countersink option, when PRO_E_HLE_ADD_CSINK equals PRO_HLE_ADD_CSINK.
PRO_E_HLE_CSINKDIAM	Stores countersink diameter. Double value. Available for countersink option, when PRO_E_HLE_ADD_CSINK PRO_HLE_ADD_CSINK.
PRO_E_HLE_DRILLDEPTH	Stores drill depth double value. Available for tapped hole, with variable depth option. That is, when PRO_E_

Element ID	Comment/Description
	HLE_STAN_TYPE equals PRO_HLE_TAPPED_TYPE, and PRO_E_HLE_DEPTH equals PRO_HLE_STD_VAR_DEPTH.
PRO_E_HLE_ADD_EXIT_CSINK	An option for Exit Counter Sink of type ProHleAddExitCsinkFlag. For exit counter sink it's value is PRO_HLE_ADD_EXIT_CSINK. For no countersink, value is PRO_HLE_NO_EXIT_CSINK. It is not available for assembly mode. In part mode will fail if entry and exit surfaces of hole are non-planar and non-parallel.
PRO_E_HLE_EXIT_CSINKANGLE	Stores exit countersink angle double value. Available for exit countersink option, that is, PRO_E_HLE_ADD_EXIT_CSINK == PRO_HLE_ADD_EXIT_CSINK.
PRO_E_HLE_EXIT_CSINKDIAM	Stores exit countersink diameter double value. Available for exit countersink option, that is, PRO_E_HLE_ADD_EXIT_CSINK == PRO_HLE_ADD_EXIT_CSINK.
PRO_E_HLE_ADD_NOTE	It is an option for add note. Of type ProHleAddNoteFlag. The default value is add note, i.e.PRO_HOLE_ADD_NOTE_FLAG For no note, the value is PRO_HOLE_NO_NOTE_FLAG.
PRO_E_HOLE_NOTE	This element is not accessible through Creo Parametric TOOLKIT. Default note will be created when PRO_E_HLE_ADD_NOTE is set to PRO_HOLE_ADD_NOTE_FLAG.
PRO_E_HLE_THRDTOSEL	This element stores reference when PRO_E_HLE_THRD_DEPTH==PRO_HLE_TO_SELECTED_THREAD.
Custom Hole	
PRO_E_HLE_TYPE_NEW	PRO_HLE_CUSTOM_TYPE
PRO_E_HLE_ADD_CBORE	The description of these items are same as described in Standard Hole section.
PRO_E_HLE_ADD_CSINK	
PRO_E_HLE_HOLEDIAM	
PRO_E_HLE_DRILLANGLE	
PRO_E_HLE_CSINKANGLE	
PRO_E_HLE_CBOREDEPTH	
PRO_E_HLE_CBOREDEPTH	
PRO_E_HLE_CSINKDIAM	
PRO_E_HLE_DRILLDEPTH	
PRO_E_HLE_DEPTH	
PRO_E_STD_HOLE_DEPTH_REF	
PRO_E_HLE_DEPTH_DIM_TYPE	
PRO_E_HLE_CRDIR_FLIP	
PRO_E_HLE_ADD_EXIT_CSINK	

Element ID	Comment/Description
PRO_E_HLE_EXIT_CSINKANGLE	
PRO_E_HLE_EXIT_CSINKDIAM	

Hole Placement Types

Creo Parametric TOOLKIT supports several placement types for holes.

Hole Placement

The elements discussed in the following sections specify how to place a hole in relation to the model geometry. The reference entity elements are carried as selection objects, and the other elements carrying actual values of distances, offsets or angles.

Creo Parametric TOOLKIT supports the following types of hole placement:

- Linear Hole on a Plane on page 951
- Radial Hole on Plane with Radial Dimensioning on page 952
- Radial Hole on Plane with Diameter Dimensioning on page 953
- Radial Hole on Plane with Linear Dimensioning on page 953
- Radial Hole on Cone or Cylinder on page 953
- Coaxial Hole with Axis as Primary Reference on page 954
- Coaxial Hole with Primary Reference not Axis on page 954
- Onpoint Hole with Primary Reference as a Point on Surface on page 954
- Onpoint Hole with Primary Reference as Datum Point on page 954
- Onpoint Hole with Primary Reference as Datum Point with Orientation References on page 955
- Hole with Primary Reference as Sketch on page 955

Linear Hole on a Plane

Linear placement requires as references either

- Two linear non-parallel edges in the plane of placement

or

- Two planar non-parallel surfaces, both normal to the plane of placement.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary selection, that is, planar surface or datum plane.
PRO_E_HLE_PL_TYPE	Set to PRO_HLE_PL_TYPE_LIN

Element ID	Comment/Description
PRO_E_HLE_DIM_REF1	First secondary selection, that is, plane, edge, or axis. If edge or axis is normal to placement plane, another selection is required for dimensioning the hole. So this may require two selections.
PRO_E_HLE_PLC_ALIGN_OPT1	- Set to PRO_HLE_PLC_ALIGN to align the hole to the reference. - Set to PRO_HLE_PLC_NOT_ALIGN to use the DIST1 reference.
PRO_E_HLE_DIM_DIST1	Distance with regard to PRO_E_HLE_DIM_REF1.
PRO_E_HLE_DIM_REF2	Second secondary selection, that is, plane, edge, or axis. - If edge or axis is normal to placement plane another selection is required for dimensioning the hole. So this may require two selections.
PRO_E_HLE_PLC_ALIGN_OPT2	- Set to PRO_HLE_PLC_ALIGN to align the hole to the reference. - Set to PRO_HLE_PLC_NOT_ALIGN to use the DIST2 reference.
PRO_E_HLE_DIM_DIST2	Distance with regard to PRO_E_HLE_DIM_REF2.
PRO_E_LIN_HOLE_DIR_REF	Uses this reference to define the direction of the placement dimension scheme. This element is available if the secondary element PRO_E_HLE_DIM_REF1 contains an axis reference normal to the current hole's primary reference.

Radial Hole on Plane with Radial Dimensioning

Locating radial holes requires the first reference to be the axis of placement. This axis is a polar placement (r-theta), where r is the radial distance from a plane, and theta is the angle with respect to a plane.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, planar surface/datum plane
PRO_E_HLE_PL_TYPE set to PRO_HLE_PL_TYPE_RAD	
PRO_E_HLE_AXIS	Axis for radial hole
PRO_E_HLE_DIM_RAD	Radial distance with regard to PRO_E_HLE_AXIS
PRO_E_HLE_REF_PLANE_1	Reference plane against which angular distance will be measured
PRO_E_HLE_REF_ANG_1	Angular distance with regard to PRO_E_HLE_REF_PLANE_1

Radial Hole on Plane with Diameter Dimensioning

Locating these holes is similar to radial holes with radial dimensioning. The difference is the distance specified is in the form of diametrical distance.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, planar surface/datum plane
PRO_E_HLE_PL_TYPE set to PRO_HLE_PL_TYPE_RAD_DIA_DIM	
PRO_E_HLE_AXIS	Axis for radial hole
PRO_E_HLE_DIM_DIA	Diameter distance with regard to PRO_E_HLE_AXIS
PRO_E_HLE_REF_PLANE_1	Reference plane against which angular distance will be measured
PRO_E_HLE_REF_ANG_1	Angular distance with regard to PRO_E_HLE_REF_PLANE_1

Radial Hole on Plane with Linear Dimensioning

This type of hole placement uses an angle with respect to a plane and a linear distance from the axis of placement.

This placement type is available when you set the configuration option radial_hole_linear_dim to “YES”.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, planar surface/datum plane
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_RAD_LIN_DIM	
PRO_E_HLE_AXIS	Axis for radial hole
PRO_E_HLE_DIM_LIN	Linear distance with regard to PRO_E_HLE_AXIS
PRO_E_HLE_REF_PLANE_1	Reference plane against which angular distance will be measured
PRO_E_HLE_REF_ANG_1	Angular distance with regard to PRO_E_HLE_REF_PLANE_1

Radial Hole on Cone or Cylinder

This hole placement type requires the selection of a cone or cylinder for primary placement.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, Cone or Cylinder
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_RAD	
PRO_E_HLE_REF_PLANE	Reference plane against which angular distance will be measured
PRO_E_HLE_REF_ANG	Angular distance with regard to PRO_E_HLE_REF_PLANE
PRO_E_HLE_NORM_PLA	Reference plane for linear measurement
PRO_E_HLE_NORM_OFFST	Distance with regard to PRO_E_HLE_NORM_PLA

Coaxial Hole with Axis as Primary Reference

Coaxial hole placement requires an axis and a placement plane to complete the placement.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, Axis
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_COAX	
PRO_E_HLE_PLCMNT_PLANE	Placement surface

Coaxial Hole with Primary Reference not Axis

This is a special case of coaxial hole in which the primary selection is a surface. The axis must be normal to the selected surface:

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, Surface
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_COAX	
PRO_E_HLE_AXIS	Axis

Onpoint Hole with Primary Reference as a Point on Surface

This placement type requires a point of type ‘On Surface Point’. The hole is placed normal to the surface on which the point was created. The hole passes through the selected point. The depth of hole is measured from the datum point.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, on Surface Created Datum Point
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_ON_PNT	

Onpoint Hole with Primary Reference as Datum Point

This placement type requires a point of type ‘Datum Point’. It also requires a surface reference. The datum point is projected on the surface. The depth of the hole is measured from the projected point.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, Datum Point
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_ON_PNT	
PRO_E_HLE_PLCMNT_PLANE	Placement surface

Hole with Primary Reference as Sketch

This placement type requires a point of type ‘Sketch’. The offset references are used as Sketch references.

Element ID	Comment/Description
PRO_E_STD_SECTION	Primary Selection, Sketch
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_SKT_DTM_PNT	
PRO-E_HOLE_SKDP_OPTIONS	Sketched datum point hole option

Onpoint Hole with Primary Reference as Datum Point with Orientation References

This placement type requires a point of type ‘Datum Point’. The direction of hole is defined using the orientation references. The hole passes through the selected point. The depth of hole is measured from the datum point.

Element ID	Comment/Description
PRO_E_HLE_PRIM_REF	Primary Selection, Datum Point
PRO_E_HLE_PL_TYPE is set to PRO_HLE_PL_TYPE_ON_PNT	
PRO_E_HLE_FT_DIR_REF	References to define the orientation of the hole.
PRO_E_HLE_FT_DIR_OPT	Option to define the direction in which the hole is oriented, perpendicular or parallel, to the orientation references.

Miscellaneous Information

The following sections discuss important issues relating to hole feature creation.

Hole Parameter Files

Hole parameter files are setup files used to build the user interface for the hole. Programmatic hole creation uses the same files. New sets of customized files can be added as required. The values assigned to the elements PRO_E_HLE_THRDSERIS, PRO_E_HLE_SCREWSIZE, and therefore PRO_E_HLE_HOLEDIAM depend on these files.

Creo Parametric and Creo Parametric TOOLKIT load the hole parameter file (*.hol) in following order:

1. Directory specified in configuration option hole_parameter_file_path
2. Current Directory
3. System hole parameter directory, that is,
4. [PROE DIR]/text/hole

Find the hole diameter from the values of PRO_E_HLE_THRDSERIS and PRO_E_HLE_SCREWSIZE specified in the *hol files. In the Creo Parametric User Interface, element PRO_E_HLE_THRDSERIS is represented as the selection between UNC, UNF or ISO.

Hole Diameter

The drill diameter PRO_E_HLE_HOLEDIAM, as required for the Standard Type of holes, must be smaller than the thread diameter calculated from the .hol file for the threaded hole. As specified in the *.hol files, the thread diameter is the element corresponding to BASIC_DIAM column and the selected screw size row in the table, as specified in the selected .hol file. If the PRO_E_HLE_HOLEDIAM is not smaller than the thread diameter, the ProFeatureCreate() function fails and returns a PRO_TK_GENERAL_ERROR.

Follow these steps to enter the proper value for PRO_E_HLE_HOLEDIAM:

1. Determine the values to pass from the Creo Parametric user interface to the following elements:
   - PRO_E_HLE_THRDSERIS. Note that UNC corresponds to 0, UNF to 1, and ISO to 2. These values change if you create a local .hol file.
   - PRO_E_HLE_SCREWSIZE (the values start with zero).
2. From the Creo Parametric User Interface, set the options to be passed to the elements PRO_E_HLE_THRDSERIS and PRO_E_HLE_SCREWSIZE. For example, ISO with M1X25 or UNC with 1-64.
3. Observe the value hole diameter in the dialog box. The dialog box appears grayed out unless you set the configuration option hole_diameter_override to yes.
4. The value thus obtained for the hole diameter should be greater than the value defined for element PRO_E_HLE_HOLEDIAM.

Order of Element Specification

Be sure to enter the elements into the element tree in the order specified by the tables in PRO_E_HLE_COM Values on page 940. Failure to follow these sequences may result in either ProFeatureCreate() failing with a PRO_TK_GENERAL_ERROR error return, or in creation of a feature which fails to get redefined.

Hole-specific Functions

Functions Introduced:

- **ProHolePropertyGet()**
- **ProElementHoleThreadSeriesGet()**
- **ProElementHoleThreadSeriesSet()**
- **ProElementHoleScrewSizeGet()**
- **ProElementHoleScrewSizeSet()**

You can use the function `ProHolePropertyGet()` to retrieve the value of the indicated hole value property. Only properties listed in the enum `ProHoleProperty` are supported.

The function `ProElementHoleThreadSeriesGet()` returns the type of thread from the hole feature element tree as a wide string.

Use the function `ProElementHoleThreadSeriesSet()` to set the type of thread in the hole feature element tree. The thread type is updated in the element `PRO_E_HLE_THRDSERIS`.

The function `ProElementHoleScrewSizeGet()` gets the size of screw from the hole feature tree elements `PRO_E_HLE_THRDSERIS` and `PRO_E_HLE_SCREWSIZE` as a wide string.

Use the function `ProElementHoleScrewSizeSet()` to set the size of screw in the hole feature element tree. The screw size is updated in the element `PRO_E_HLE_SCREWSIZE`.

Note

The screw size depends on the type of thread. Therefore, before you call the function `ProElementHoleScrewSizeSet()` you must ensure that the thread type is set in the element `PRO_E_HLE_THRDSERIS`.

41

Element Trees: Shell

Thischapter introduces and shows how to create, redefine and access Shell features in Creo Parametric TOOLKIT.

Introduction to Shell Feature

When Creo Parametric makes a shell, all features that were added to the solid before creating the Shell feature are hollowed. Therefore, the order of feature creation is very important when you use the Shell feature.

The Shell feature hollows the inside of the solid, leaving a shell of a specified wall thickness. It allows you to remove a surface or surfaces from the shell. If you do not select a surface to remove, a “closed shell” is created, with the inside of the part completely hallowed out and no access to the hollow. When defining a shell, you can also select surfaces with different thickness values. On flipping the thickness side, the shell thickness is added to the outside of the part.

You can also shell surfaces that are tangent to their neighbors at one or more boundaries. At the tangent edge where the separation of the shell offset occurs, a normal capping surface is constructed to close the gap.

You can also exclude one or more surfaces from being shelled. This process is called partial shelling. There are two different algorithms for partial shelling – one for concave corner surfaces and the other for convex corner surfaces. These algorithms prevent the shell subtraction volume from penetrating through the solid. In case of a part where both concave and convex corner surfaces are to be excluded, the exclusion can be achieved in multiple partial shells, each using different algorithms.

The following are the restrictions on Shell feature creation:

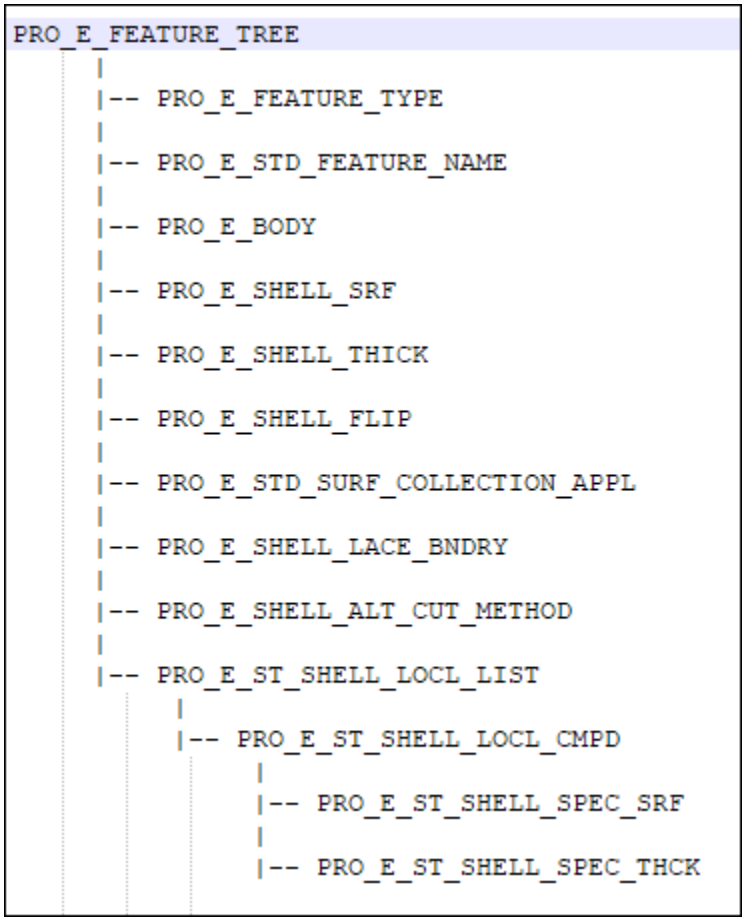
- You cannot add shells to any part that has a surface that moves from tangency to a point.
- You cannot select a surface to be removed that has a vertex created by the intersection of three curved surfaces.
- The surface to be removed must be surrounded by edges (a fully revolved surface of revolution is not valid) and the surfaces that intersect the edge must form an angle of less than 180 degrees through the solid geometry. As long as this condition is met, you can select any sculpted surface as the surface to be removed.
- When you select surfaces that have other surfaces tangent to them for independent thickness, all surfaces that are tangent must have the same thickness. Otherwise, the Shell feature fails.

  For example, if you shell a part that contains a hole and you want the thickness of the hole wall to be different from the overall thickness, you must select both surfaces (cylinders) that make up the hole and offset them at the same distance.
- By default, a shell creates geometry with a constant wall thickness. If the system cannot create a constant thickness, the Shell feature fails.

Feature Element Tree for the Shell Feature

The element tree for the Shell feature is documented in the header file `ProShell.h`, and has a simple structure. The following figure demonstrates the element tree structure:

Feature Element Tree for Shell Feature



The shell element tree contains standard element types. The following list details special information about the elements in this tree:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should have the value `PRO_FEAT_SHELL`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the Shell feature.
- `PRO_E_BODY`—Specifies the body options and is a branch of the general body options elements defined in the `ProBodyOpts.h` as follows:
  - `PRO_E_BODY_USE`—The valid value is `PRO_BODY_USE_SELECTED`

- PRO_E_BODY_SELECTED—Must contain a single selected body to shell.

- PRO_E_SHELL_SRF— Specifies the selected surfaces to be removed from the part to create the Shell feature. This element is optional.
- PRO_E_SHELL_THICK—Specifies the default thickness for the shell. It must be positive and greater than zero.
- PRO_E_SHELL_FLIP—Specifies the side of the shell to be flipped and has the following values:
  - PRO_SHELL_OUTSIDE
  - PRO_SHELL_INSIDE
- PRO_E_STD_SURF_COLLECTION_APPL—Specifies the surfaces to be excluded during the Shell feature creation.
- PRO_E_SHELL_LACE_BNDRY—Specifies the lace boundary that forms the closure of excluded surfaces or inner surfaces. It has the following values:
  - PRO_SHELL_LACE
  - PRO_SHELL_DONT_LACE
- PRO_E_SHELL_ALT_CUT_METHOD—Specifies the alternate cut method used for partial shell volume subtraction. It has the following values:
  - PRO_SHELL_ALT_CUT_METHOD_NO—Specifies the algorithm for concave corner surfaces.
  - PRO_SHELL_ALT_CUT_METHOD_YES—Specifies the algorithm for convex corner surfaces.
- PRO_E_ST_SHELL_LOCL_LIST—Specifies an array of local thickness of the type PRO_E_ST_SHELL_LOCL_CMPD which consists of the following elements:
  - PRO_E_ST_SHELL_SPEC_SRF—Specifies the surface selected for specifying the local thickness value. This surface cannot be one of the surfaces selected to be removed.
  - PRO_E_ST_SHELL_SPEC_THCK—Specifies the local thickness value of the selected surface (initially equal to default shell thickness). It must be positive and greater than zero.

Creating a Shell Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Shell feature based on the element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 of chapter Element Trees: Principles of Feature Creation on page 764.

Redefining a Shell Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Shell feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 of chapter Element Trees: Principles of Feature Creation on page 764.

Accessing a Shell Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Shell feature and to retrieve the element tree description of a Shell feature. For more information about `ProFeatureElemtreeExtract()` refer to the section Feature Inquiry on page 785 of chapter Element Trees: Principles of Feature Creation on page 764.

42

Element Trees: Patterns

This chapter describes the element tree structure required to create patterns of features. The chapter on Element Trees: Principles of Feature Creation on page 764 is a necessary background for this topic. Read that chapter before this one.

Introduction

Using Creo Parametric TOOLKIT, you can create patterns of features, including those not supported by Creo Parametric TOOLKIT feature creation. Consequently, you can programmatically create patterns of any feature that can be patterned in Creo Parametric.

The creation and manipulation of patterns use the following Creo Parametric TOOLKIT objects:

- `ProPattern`—A structure that contains the type and owner of the pattern, and an opaque pattern handle
- `ProPatternClass`—An enumerated type that contains the pattern class, which specifies either a feature pattern (`PRO_FEAT_PATTERN`) or a group pattern (`PRO_GROUP_PATTERN`)

The procedure for creating a pattern is similar to creating features, in that you construct an element tree and pass this element tree to Creo Parametric. When you pass the tree to Creo Parametric, you also specify the feature to be patterned.

The Element Tree for Pattern Creation

Unlike the element tree for features, the element tree for a pattern does not contain information about the construction of new features. Rather, the element tree contains information needed to make copies of existing features at specified locations on the model. For example, the element tree for a pattern of holes does not contain the geometry (such as edges) used to place the holes, but contains the dimensions and dimension variations used to pattern the specified hole.

You construct the element tree for a pattern by following the procedure described in chapter Element Trees: Principles of Feature Creation on page 764:

1. Allocate tree elements using the function `ProElementAlloc()`.
2. Set values of the elements using the function `ProElementValueSet()`.
3. Add elements to the tree using `ProElemtreeElementAdd()`.

As with feature creation, the system cannot create your pattern unless the element tree is correct.

The element tree for a pattern is documented in the header file `ProPattern.h`. This tree contains the same information required when you create a pattern in an interactive session of Creo Parametric. Therefore, you should be familiar with how to create a pattern interactively before you try to understand the element tree.

Note

It is highly recommended that you use the new element tree from `ProPattern.h`. The old element tree is only for your reference.

The following figure shows a part of the element tree for patterns. All elements are described in detail in the following sections.

A Part of the Element Tree for Patterns

```
PRO_E_PATTERN_ROOT
|
|--- PRO_E_GENPAT_TYPE
|
|--- PRO_E_GENPAT_REGEN_METHOD
|
|--- PRO_E_STD_FEATURE_NAME
|
|       * Dim Pattern *
|--- PRO_E_GENPAT_DIM
|
|       * Table Pattern *
|--- PRO_E_GENPAT_TABLE
|
|       * Ref Pattern *
|--- PRO_E_GENPAT_REF
|
|       * Fill Pattern *
|--- PRO_E_GENPAT_FILL
|
|       * Curve Pattern *
|--- PRO_E_GENPAT_CURVE
|
|       * Dir Pattern *
|--- PRO_E_GENPAT_DIR
|
|       * Axis Pattern *
|--- PRO_E_GENPAT_AXIS
|
|       * Point Pattern *
|--- PRO_E_GENPAT_POINT
|
|--- PRO_E_STD_SECTION
|
|--- PRO_E_PAT_GEOM_REFS
|
|-- PRO_E_FLEX_OPTS_CMPND
|
|--- PRO_E_PAT_APPLICATIONS
```

The element with the identifier PRO_E_GENPAT_TYPE sets the type of the pattern to be created. The structure of the rest of the element tree depends strongly on the value of this element. Valid values for the PRO_E_GENPAT_TYPE element are as follows:

- PRO_GENPAT_REF_DRIVEN—Reference pattern
- PRO_GENPAT_DIM_DRIVEN—Dimension pattern
- PRO_GENPAT_TABLE_DRIVEN—Table pattern
- PRO_GENPAT_FILL_DRIVEN—Fill pattern
- PRO_GENPAT_DIR_DRIVEN—Direction pattern
- PRO_GENPAT_AXIS_DRIVEN—Axis pattern
- PRO_GENPAT_POINT_DRIVEN—Point pattern
- PRO_GENPAT_CRV_DRIVEN—Curve pattern

The element with the identifier PRO_E_GENPAT_REGEN_METHOD sets the regeneration method for the pattern. The regeneration method varies with the complexity of the pattern. Valid values for the PRO_E_GENPAT_REGEN_METHOD element are as follows:

- PRO_PAT_GENERAL—General pattern. This is the most complex type of pattern.
- PRO_PAT_VARYING—Varying pattern.
- PRO_PAT_IDENTICAL—Identical pattern. This is the least complex type of pattern.

For more information on regeneration methods, see the *Part Modeling User's Guide*.

You must populate the element PRO_E_STD_SECTION with a valid reference sketch and other related elements for the following pattern types:

- a point pattern that uses an element of type PRO_GENPAT_REF_SKETCH
- a fill pattern
- a curve pattern

For more information on how to populate the section elements with the valid sketched reference, refer to the Element Trees: Sketched Features on page 1004 chapter.

The following sections describe the types of pattern in more detail.

Reference Patterns

A reference pattern uses an existing pattern as a guide for the placement of the new pattern members. Consequently, if the pattern type is `PRO_GENPAT_REF_DRIVEN`, the element tree requires only that you specify the type of the reference pattern.

Element Tree for a Reference Pattern

```
      * Ref Pattern *

|--- PRO_E_GENPAT_REF
|       |
|       |--- PRO_E_GENPAT_REF_TYPE
```

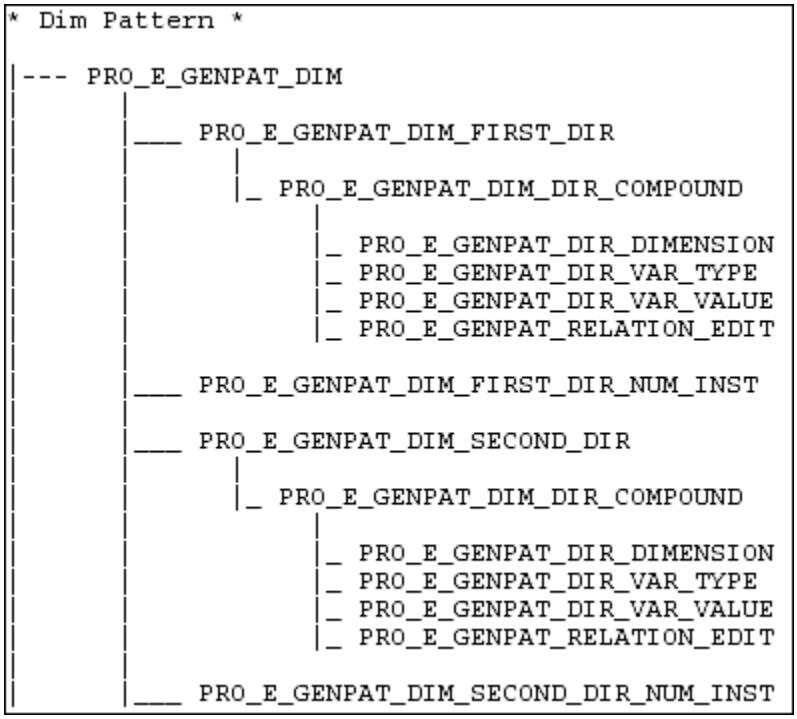
The element with identifier `PRO_E_GENPAT_REF` specifies the type of pattern to be created. The valid values are as follows:

- `PRO_PAT_FEATURE`—Use feature pattern references.
- `PRO_PAT_GROUP`—Use group pattern references.
- `PRO_PAT_BOTH`—Use feature and group pattern references.

Dimension Patterns

If the pattern type is `PRO_GENPAT_DIM_DRIVEN`, the element tree must include information about the dimensions used to drive the pattern. You must specify this information for each direction in which the feature is to be patterned.

Element Tree for a Dimension Pattern



The elements with identifiers PRO_E_GENPAT_DIM_FIRST_DIR and PRO_E_GENPAT_DIM_SECOND_DIR contain information about the pattern dimensions.

These elements are array elements that contain as many PRO_E_GENPAT_DIM_DIR_COMPOUND elements as are required to complete the pattern. The following table describes the contents of the PRO_E_GENPAT_DIM_DIR_COMPOUND element.

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_GENPAT_DIR_DIMENSION	Dimension	PRO_VALUE_TYPE_SELECTION	
PRO_E_GENPAT_DIR_VAR_TYPE	Variation type	PRO_VALUE_TYPE_INT	PRO_PAT_RELATION_DRIVEN, PRO_PAT_VALUE_DRIVEN
PRO_E_GENPAT_DIR_VAR_VALUE	Variation value (increment)	PRO_VALUE_TYPE_DOUBLE	
PRO_E_GENPAT_RELATION_EDIT	Relation	Application (PRO_VALUE_TYPE_POINTER)	

The element PRO_E_GENPAT_DIR_VAR_TYPE specifies whether the pattern increment is relation-driven or value-driven. If the increment is relation-driven, the element PRO_E_GENPAT_RELATION_EDIT contains an array of wide strings whose members are individual relations.

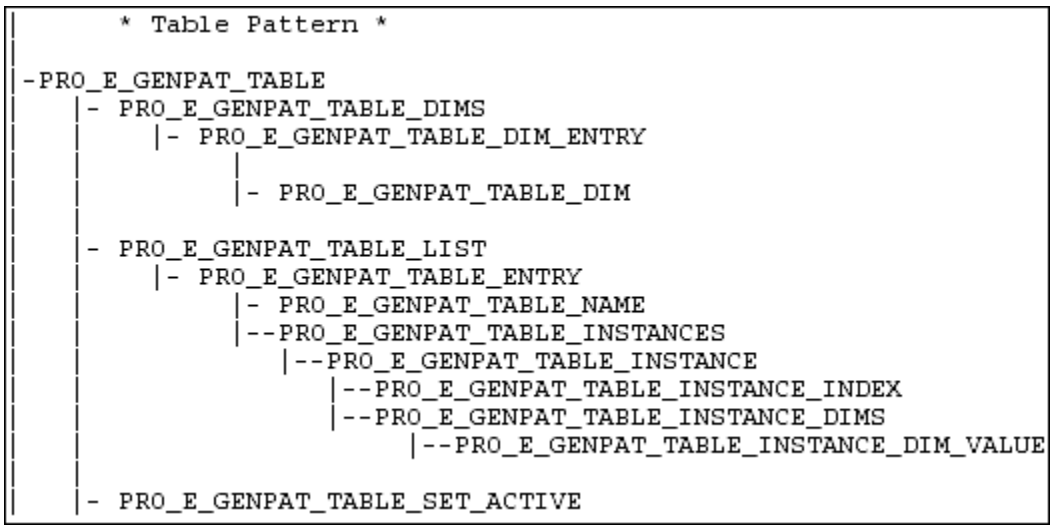
The elements PRO_E_GENPAT_DIM_FIRST_DIR_NUM_INST and PRO_E_PAT_SECOND_DIR_NUM_INST specify the number of instances in each of the pattern dimensions.

Even if the pattern extends in only one direction, you must specify elements for the second direction. In this case, add an empty PRO_E_PAT_SECOND_DIR element and set the value of the PRO_E_PAT_SECOND_DIR_NUM_INST element to 1 (not 0).

Table Patterns

If the pattern type is PRO_GENPAT_TABLE_DRIVEN, your element tree must contain the table-driven dimensions and table information (variation in dimensions for each instance). The following figure shows the elements of the Table pattern:

Element Tree for a Table Pattern



The PRO_E_GENPAT_TABLE_DIMS element is an array that contains one table dimension (PRO_E_GENPAT_TABLE_DIM) element for each dimension to be varied in the tables. The value of each PRO_E_GENPAT_TABLE_DIM element is a ProSelection object for the corresponding dimension.

The PRO_E_GENPAT_TABLE_LIST element is an array element that contains all the tables that control the pattern. This element should contain one PRO_E_GENPAT_TABLE_ENTRY element for each table.

Each PRO_E_GENPAT_TABLE_ENTRY element contains the name of the table (PRO_E_GENPAT_TABLE_NAME) and table instances (PRO_E_GENPAT_TABLE_INSTANCES).

Each PRO_E_GENPAT_TABLE_INSTANCE element contains an index number (PRO_E_GENPAT_TABLE_INSTANCE_INDEX) element and a dimensions (PRO_E_GENPAT_TABLE_INSTANCE_DIMS) element. The PRO_E_GENPAT_TABLE_INSTANCE_DIMS element is an array element that must contain one dimension value (PRO_E_GENPAT_TABLE_INSTANCE_DIM_VALUE) element for each of the selected dimensions in the PRO_E_PAT_MULT_TABLE_DIMS element. Note that the dimension value specifies the value of the selected dimension, not the dimension increment.

The following table lists the contents of each PRO_E_GENPAT_TABLE_ENTRY element.

Element ID Values	Element Name	Data Type
PRO_E_GENPAT_TABLE_NAME	Table name	PRO_VALUE_TYPE_WSTRING
PRO_E_GENPAT_TABLE_INSTANCES	Table instances	Array
PRO_E_GENPAT_TABLE_INSTANCE	Table instance	Compound
PRO_E_GENPAT_TABLE_INSTANCE_INDEX	Instance index	PRO_VALUE_TYPE_INT
PRO_E_GENPAT_TABLE_INSTANCE_DIMS	Dimension variations	Compound
PRO_E_GENPAT_TABLE_INSTANCE_DIM_VALUE	Dimension value	PRO_VALUE_TYPE_DOUBLE

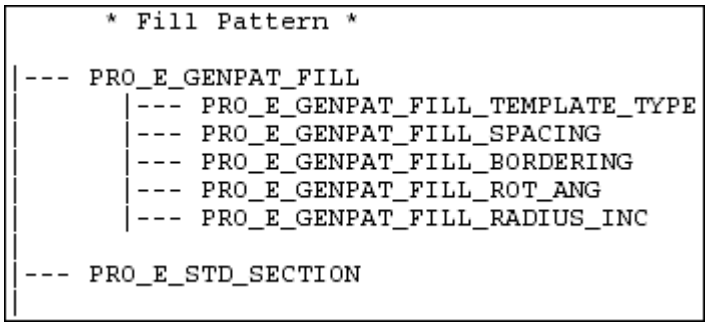
The element PRO_E_GENPAT_TABLE_SET_ACTIVE sets the active table for the pattern. Valid values are 0 (for the first table) through (num_tables - 1), where num_tables is the number of tables in the element tree.

Fill Patterns

A fill pattern controls the pattern by filling an area with pattern members. You can select a grid to define this area.

If the pattern type in your element tree is PRO_GENPAT_FILL_DRIVEN, the element tree must contain information about the grid and pattern members. The following figure shows the elements of the Fill pattern:

Element Tree for a Fill Pattern



The element PRO_E_GENPAT_FILL_TEMPLATE_TYPE specifies the type of grid template that you want to use to create a fill pattern.

The element PRO_E_GENPAT_FILL_SPACING specifies the spacing between the pattern members.

The element PRO_E_GENPAT_FILL_BORDERING specifies the minimum distance between the centers of the pattern members and the area boundary.

The element PRO_E_GENPAT_FILL_ROT_ANG specifies the rotation angle of the grid about the Csys origin.

The element PRO_E_GENPAT_FILL_RADIUS_INC specifies the radial spacing for circular and spiral grids.

The following table lists the contents of each PRO_E_GENPAT_FILL element.

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_GENPAT_FILL_TEMPLATE_TYPE	Fill template	PRO_VALUE_TYPE_INT	PRO_GENPAT_SQUARE_FILL, PRO_GENPAT_DIAMOND_FILL, PRO_GENPAT_TRIANGLE_FILL, PRO_GENPAT_CIRCLE_FILL, PRO_GENPAT_CURVE_FILL, PRO_GENPAT_SPIRAL_FILL
PRO_E_GENPAT_FILL_SPACING	Fill spacing	PRO_VALUE_TYPE_DOUBLE	
PRO_E_GENPAT_FILL_BORDERING	Fill bordering	PRO_VALUE_TYPE_DOUBLE	
PRO_E_GENPAT_FILL_ROT_ANG	Fill angle	PRO_VALUE_TYPE_DOUBLE	
PRO_E_GENPAT_FILL_RADIUS_INC	Fill radius increment	PRO_VALUE_TYPE_DOUBLE	

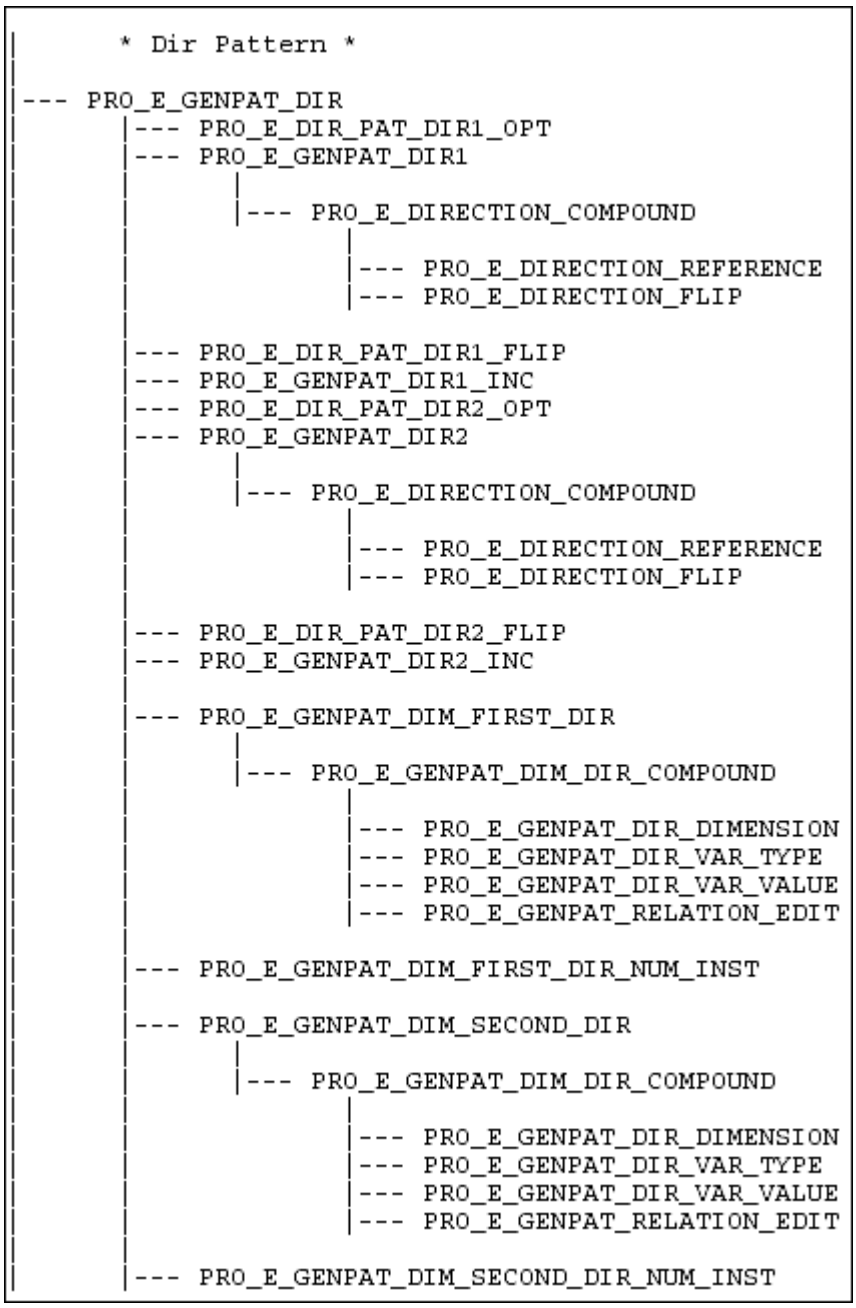
Direction Patterns

A direction pattern adds pattern members in one or two selected directions.

If the pattern type in the element tree is PRO_GENPAT_DIR_DRIVEN, the element tree must contain the information about the two directions and the pattern members.

The following figure shows the elements of a direction pattern:

Element Tree for a Direction Pattern



For a direction patter, use the elements `PRO_E_DIR_PAT_DIR1_OPT` and `PRO_E_DIR_PAT_DIR2_OPT` to specify the pattern orientation in the first and second direction, respectively. The orientation options are translation, rotation, and coordinate system.

Depending on the selected orientation, choose references for the two directions using the element `PRO_E_DIRECTION_REFERENCE`. To flip the selected directions, use the elements `PRO_E_DIR_PAT_DIR1_FLIP` and `PRO_E_DIR_`

PAT_DIR2_FLIP. The values of the references and the choice to flip the direction are stored in the elements PRO_E_GENPAT_DIR1 and PRO_E_GENPAT_DIR2.

The elements PRO_E_GENPAT_DIR1_INC and PRO_E_GENPAT_DIR2_INC specify the spacing between the pattern members in the first and second directions, respectively.

The elements with identifiers PRO_E_GENPAT_DIM_FIRST_DIR and PRO_E_GENPAT_DIM_SECOND_DIR contain dimension information for the pattern members in the first and second direction, respectively. These elements are array elements that contain as many PRO_E_GENPAT_DIM_DIR_COMPOUND elements as required to complete the pattern. For more information on the elements PRO_E_GENPAT_DIR_DIM_COMPOUND, PRO_E_GENPAT_FIRST_DIR_NUM_INST, and PRO_E_GENPAT_SECOND_DIR_NUM_INST, refer to the section on Dimension Patterns on page 968.

The following table lists the contents of each PRO_E_GENPAT_DIR element.

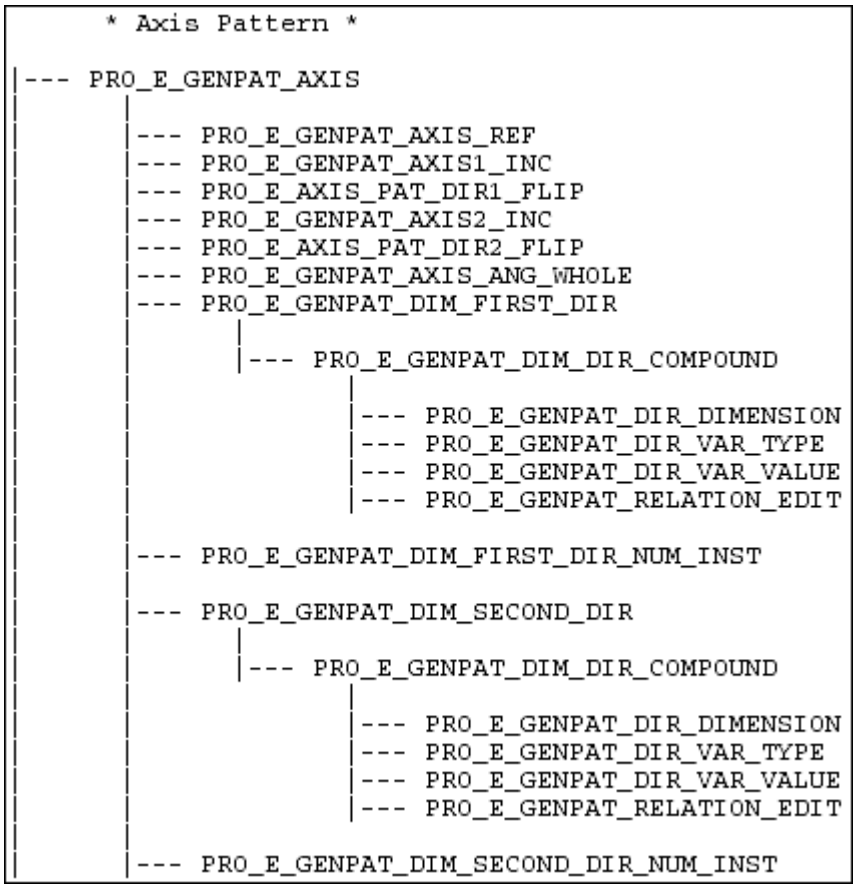
Element ID Values	Element Name	Data Type	Valid Values
PRO_E_DIR_PAT_DIR1_OPT	Direction 1st option	PRO_VALUE_TYPE_INT	PRO_GENPAT_TRANSLATIONAL, PRO_GENPAT_DIR1_ROTATIONAL
PRO_E_DIR_PAT_DIR2_OPT	Direction 2nd option	PRO_VALUE_TYPE_INT	PRO_GENPAT_TRANSLATIONAL, PRO_GENPAT_DIR2_ROTATIONAL
PRO_E_GENPAT_DIR1 and PRO_E_GENPAT_DIR2	1st direction and 2nd direction	Compound	
PRO_E_DIRECTION_COMPOUND	PRO_E_DIRECTION_COMPOUND	Compound	
PRO_E_DIRECTION_REFERENCE	Direction reference	PRO_VALUE_TYPE_SELECTION	
PRO_E_DIRECTION_FLIP	Direction flip	PRO_VALUE_TYPE_INT	Value ignored
PRO_E_DIR_PAT_DIR1_FLIP and PRO_E_DIR_PAT_DIR2_FLIP	1st direction flip and 2nd direction flip	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_GENPAT_DIR1_INC and PRO_E_GENPAT_DIR2_INC	1st direction increment and 2nd direction increment	PRO_VALUE_TYPE_DOUBLE	
PRO_E_GENPAT_DIM_FIRST_DIR and PRO_E_GENPAT_DIM_SECOND_DIR	1st direction dimensions	Array	

Axis Patterns

Use the axis pattern to create a pattern by revolving a feature around a selected axis. An axis pattern allows you to place members in the first direction or angular direction and in the second direction or radial direction.

If the pattern type in your element tree is `PRO_GENPAT_AXIS_DRIVEN`, the element tree must contain information about the pattern members in the two directions and the axis around which you want to create a pattern.

Element Tree for an Axis Pattern



The element `PRO_E_GENPAT_AXIS_REF` specifies the axis around which you want to create the pattern. The value of this element is `PRO_VALUE_TYPE_SELECTION`, which is of type `ProSelection`.

The element `PRO_E_GENPAT_AXIS1_INC` specifies the spacing between the pattern members in the first direction and is of type `PRO_VALUE_TYPE_DOUBLE`. The first direction being angular, this distance is the angular distance and the range is `-360` through `+360`.

The element PRO_E_GENPAT_AXIS2_INC specifies the spacing between the pattern members in the second direction and is of type PRO_VALUE_TYPE_DOUBLE. The second direction being radial, this distance is the linear distance and the range is -999999999.9999 through +999999999.9999.

The elements PRO_E_AXIS_PAT_DIR1_FLIP and PRO_E_AXIS_PAT_DIR2_FLIP flip the pattern members in the first and second direction, respectively, around the axis. These elements are of type PRO_VALUE_TYPE_INT.

The element PRO_E_GENPAT_AXIS_ANG_WHOLE specifies the angular extent of the pattern members and is of type PRO_VALUE_TYPE_DOUBLE. The range for this element is 0.0000 through 999999999.9999.

The value of each PRO_E_GENPAT_DIR_DIMENSION element is a ProSelection object for the corresponding dimensions.

The elements with identifiers PRO_E_GENPAT_DIM_FIRST_DIR and PRO_E_GENPAT_DIM_SECOND_DIR contain dimension information for the pattern members in the first and second direction, respectively. For more information on these elements, refer to the section on Dimension Patterns on page 968.

Curve Patterns

A curve pattern creates instances of a feature along a sketched curve or a datum curve.

If the pattern type in your element tree is PRO_GENPAT_CRV_DRIVEN, the element tree must contain information about the curve and the pattern members.

Element Tree for a Curve Pattern

```
* Curve Pattern *

|--- PRO_E_GENPAT_CURVE
|       |--- PRO_E_GENPAT_CRV_PNT_REF
|       |--- PRO_E_GENPAT_CRV_PLC_TYPE
|       |--- PRO_E_GENPAT_CRV_SPACING
|       |--- PRO_E_GENPAT_CRV_NUMBER
|       |--- PRO_E_GENPAT_CRV_FLIP
|
|--- PRO_E_STD_SECTION
|
```

The element PRO_E_GENPAT_CRV_PNT_REF specifies the curve to be used as a reference.

After you select the reference, you can either select a sketched curve or draw a curve using Sketcher. The element PRO_E_GENPAT_CRV_PLC_TYPE specifies the curve types.

The element PRO_E_GENPAT_CRV_SPACING specifies the separation between the pattern members.

The element PRO_E_GENPAT_CRV_NUMBER specifies the number of pattern members to be created.

The element PRO_E_GENPAT_CRV_FLIP specifies flipping the curve used in patterning.

The following table lists the contents of each PRO_E_GENPAT_CRV element.

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_GENPAT_CRV_PNT_REF	Curve reference	PRO_VALUE_TYPE_SELECTION	
PRO_E_GENPAT_CRV_PLC_TYPE	Curve type	PRO_VALUE_TYPE_INT	
PRO_E_GENPAT_CRV_SPACING	Curve spacing	PRO_VALUE_TYPE_DOUBLE	0.0000 to 1000000.0000
PRO_E_GENPAT_CRV_NUMBER	Curve number	PRO_VALUE_TYPE_INT	
PRO_E_GENPAT_CRV_FLIP	Curve flip	PRO_VALUE_TYPE_INT	0 or 1

Point Patterns

A point pattern creates a pattern by placing a pattern member at a particular point.

The following figure shows the elements of a point pattern:

Element Tree for a Point Pattern

```
* Point Pattern *

|--- PRO_E_GENPAT_POINT
|       |--- PRO_E_GENPAT_POINT_REF_TYPE
|       |--- PRO_E_GENPAT_POINT_REF
|
|--- PRO_E_STD_SECTION
|
```

If the pattern type in the element tree is PRO_GENPAT_POINT_DRIVEN, the element tree must contain the information about the reference point and the actual point at which you want to draw the pattern.

Use the element PRO_E_GENPAT_POINT_REF_TYPE to select the type of the point at which you want to repeat the selected feature. This selected point is the reference point for creating the pattern. You can select a point from the following options:

- Internal or External sketch
- Datum point feature

After the type of the reference point is set, use the element PRO_E_GENPAT_POINT_REF to select the actual point.

The following table lists the contents of each PRO_E_GENPAT_POINT element.

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_GENPAT_POINT_REF_TYPE	Point type	PRO_VALUE_TYPE_INT	PRO_GENPAT_REF_SKETCH, PRO_GENPAT_REF_POINT
PRO_E_GENPAT_POINT_REF	Point reference	PRO_VALUE_TYPE_SELECTION	

Selecting References for Pattern

The compound element PRO_E_PAT_GEOM_REFS contains information about the references selected for the pattern leader.

The element PRO_E_PAT_GEOM_REFS allows you to only read the references selected for pattern leader. You cannot select the references in Creo Parametric TOOLKIT using these elements.

The following figure shows the element tree for selecting references for pattern leader:

```
|--- PRO_E_PAT_GEOM_REFS
|       |
|       |_ PRO_E_STD_SURF_COLLECTION_APPL
|       |
|       |_ PRO_E_PAT_CRV_DTM_REFS
|
```

The compound element PRO_E_PAT_GEOM_REFS contains the following elements:

- PRO_E_STD_SURF_COLLECTION_APPL—Specifies the collection of surfaces that define the leader of the geometry pattern.
- PRO_E_PAT_CRV_DTM_REFS—Specifies the collection of curves and datums that define the leader of the geometry pattern.

The following table lists the contents of PRO_E_PAT_GEOM_REFS element:

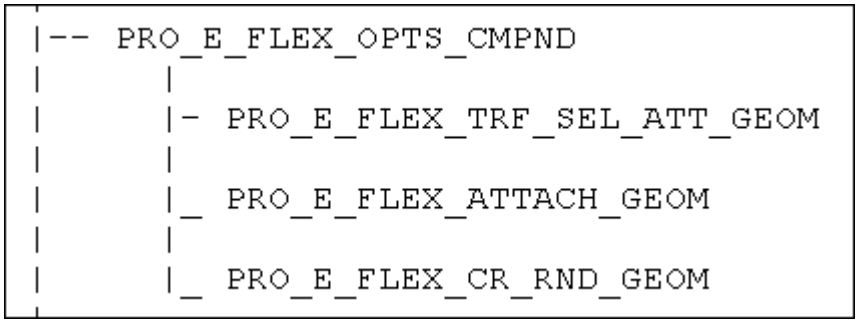
Element ID Values	Element Name	Data Type
PRO_E_STD_SURF_COLLECTION_APPL	Reference Surfaces	PRO_VALUE_TYPE_SELECTION
PRO_E_PAT_CRV_DTM_REFS	Reference Curves and Datums	PRO_VALUE_TYPE_SELECTION

Attachment Options for Pattern

The compound element PRO_E_FLEX_OPTS_CMPND contains information about the attachment options for the pattern.

The element PRO_E_FLEX_OPTS_CMPND allows you to only read the attachment options selected for a pattern. You cannot set the attachment option in Creo Parametric TOOLKIT using these elements.

The following figure shows the element tree for attachment options for the pattern:



The compound element PRO_E_FLEX_OPTS_CMPND contains the following elements:

- PRO_E_FLEX_TRF_SEL_ATT_GEOM—Specifies if the selected rounds and chamfers that attach the patterned geometry to the model must be patterned. 1 specifies that the rounds and chamfers are patterned. When you specify 0, the selected attaching rounds and chamfers are removed.
- PRO_E_FLEX_ATTACH_GEOM—Specifies if the geometry of the pattern members must be reattached to the model after patterning. 1 specifies that the pattern geometry is attached to the model.
- PRO_E_FLEX_CR_RND_GEOM—Specifies if the round or chamfer geometry of the pattern members must be recreated after patterning. 1 specifies that the pattern geometry is recreated with rounds or chamfers.

The following table lists the contents of PRO_E_FLEX_OPTS_CMPND element:

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_FLEX_TRF_SEL_ATT_GEOM	Transform selected attachment geometry	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_FLEX_ATTACH_GEOM	Attachment option	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_FLEX_CR_RND_GEOM	Round option	PRO_VALUE_TYPE_INT	0 or 1

Pattern Features

The compound element PRO_E_PAT_APPLICATIONS contains elements for pattern applications. You can create a NC sequence using the elements under the compound element PRO_E_PAT_MFG_TOOL_PATH. You can view the various parameters of a pattern using the Pattern Recognition feature elements under the compound element PRO_E_PAT_GPRF.

NC Sequence Pattern

You can create an NC sequence by using the element tree for pattern applications. The compound element PRO_E_PAT_MFG_TOOL_PATH described in this section allows you to set various options related to manufacturing order and fixtures used to pattern an NC sequence.

The following figure shows the element tree for pattern applications.

Pattern Element Tree for NC Sequence

```
|
|--- PRO_E_PAT_APPLICATIONS
|       |
|       |-- PRO_E_PAT_MFG_TOOL_PATH
|       |       |
|       |       |-- PRO_E_PAT_MFG_ORD_LEADER
|       |       |-- PRO_E_PAT_MFG_ORDER_OPT
|       |       |-- PRO_E_PAT_MFG_ORD_REVERSE
|       |       |-- PRO_E_PAT_MFG_ORD_ALT_ROWS
|       |       |-- PRO_E_PAT_MFG_ORD_ALT_DIR
|       |       |-- PRO_E_PAT_MFG_ORD_SHARED
|       |       |-- PRO_E_PAT_MFG_FIX_OFFSET
|       |       |-- PRO_E_PAT_MFG_FIX_OFF_INIT
|       |       |-- PRO_E_PAT_MFG_FIX_OFF_INCR
|       |       |-- PRO_E_PAT_MFG_SUB_OUTPUT
|       |       |-- PRO_E_PAT_MFG_SUB_OUT_MODE
|       |       |-- PRO_E_PAT_MFG_SUB_OUT_MULTAX
|       |       |-- PRO_E_PAT_MFG_SUB_OUT_COPYCL
|       |
|       |-- PRO_E_PAT_GPRF
|       |       |
```

The compound element PRO_E_PAT_MFG_TOOL_PATH contains the following elements of type integer:

- PRO_E_PAT_MFG_ORD_LEADER—Specifies the number of the pattern member that you want to use as the manufacturing leader. The default value of this element is zero, which indicates that the pattern leader itself is the manufacturing leader.
- PRO_E_PAT_MFG_ORDER_OPT—Specifies the criteria for selecting the manufacturing order. The valid values are:
  - 1— Specifies the manufacturing order based on the pattern order. Set the values of the following elements as:
    - PRO_E_PAT_MFG_ORD_REVERSE—Specify 1 to reverse the order of the pattern for the CL output.

- PRO_E_PAT_MFG_ORD_ALT_ROWS—Specify 1 to set the alternate rows of the pattern in the same direction for the CL output. Here the first and the second rows are in opposite directions with the first row being in the direction of the pattern. The manufacturing leader is the first tool path in the CL output.
- PRO_E_PAT_MFG_ORD_ALT_DIR—Specify 1 to generate the CL output with the direction of the first row being treated as the direction of the second row and the direction of second row treated as the direction of the first row until the tool paths for all the pattern members are generated. The manufacturing leader is the first tool path in the CL output.

○ 2—Specifies the manufacturing order based on the shortest distance between the pattern members.

○ 3—Select the manufacturing order for each of the pattern member from the pattern UI.

Note

The functionality to select the manufacturing order for each pattern member is currently not supported through Creo Parametric TOOLKIT.

- PRO_E_PAT_MFG_ORD_SHARED—Specify 1 to sequentially set the orders for the 4-axis or 5-axis tool paths with a common Z-axis orientation.
- PRO_E_PAT_MFG_FIX_OFFSET—Specifies the parameters for the fixture offset. Specify 1 to set the following fixture options:
  - PRO_E_PAT_MFG_FIX_OFF_INIT—Specifies the initial value of the fixture Offset.
  - PRO_E_PAT_MFG_FIX_OFF_INCR—Specifies the increment value for the fixture offset.
- PRO_E_PAT_MFG_SUB_OUTPUT—Specifies if subroutine pattern must be created. Subroutines enable you to create NC sequences, place them as macros at the beginning of the CL file, and then call them from the main body of the CL file as many times as needed.
- PRO_E_PAT_MFG_SUB_OUT_MODE—Specifies the output mode for the CL data for the subroutine. Pass the value 1 for absolute mode and 2 for incremental mode.
- PRO_E_PAT_MFG_SUB_OUT_MULTAX—Specifies if the Multax mode must be selected. Multax is related to cutter location output format where it puts the post-processor in the multi-axis output mode to process the i, j, k vector. When

in multi-axis output mode, Creo NC outputs the i, j, k vector even when the tool is in 0, 0, 1 orientation.

In Multax mode, the system will output transformed CL data rather than outputting rotate table commands.

- PRO_E_PAT_MFG_SUB_OUT_COPYCL—Specifies if the subroutine pattern definitions in CL output must be temporarily suppressed. The system will output CL data without the subroutine definitions and calls.

  Refer to Creo NC online Help for more information on subroutines.

The following table lists the contents of each PRO_E_PAT_MFG_TOOL_PATH element.

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_PAT_MFG_ORD_LEADER	Number of the leader	PRO_VALUE_TYPE_INT	0 <= value < number of instances
PRO_E_PAT_MFG_ORDER_OPT	Order options	PRO_VALUE_TYPE_INT	1, 2, 3 (if you specify 3, order may be selected only through Creo Parametric UI)
PRO_E_PAT_MFG_ORD_REVERSE	Reverse option	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_MFG_ORD_ALT_ROWS	Alternate rows	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_MFG_ORD_ALT_DIR	Alternate direction	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_MFG_ORD_SHARED	Shared orientation	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_MFG_FIX_OFFSET	Fixture offsets	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_MFG_FIX_OFF_INIT	Initial fixture offsets	PRO_VALUE_TYPE_INT	1 <= value
PRO_E_PAT_MFG_FIX_OFF_INCR	Fixture offsets increment	PRO_VALUE_TYPE_INT	1 <= value
PRO_E_PAT_MFG_SUB_OUTPUT	Subroutine option	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_MFG_SUB_OUT_MODE	Subroutine mode	PRO_VALUE_TYPE_INT	1 or 2
PRO_E_PAT_MFG_SUB_OUT_MULTAX	Subroutine multax	PRO_VALUE_TYPE_INT	0 or 1

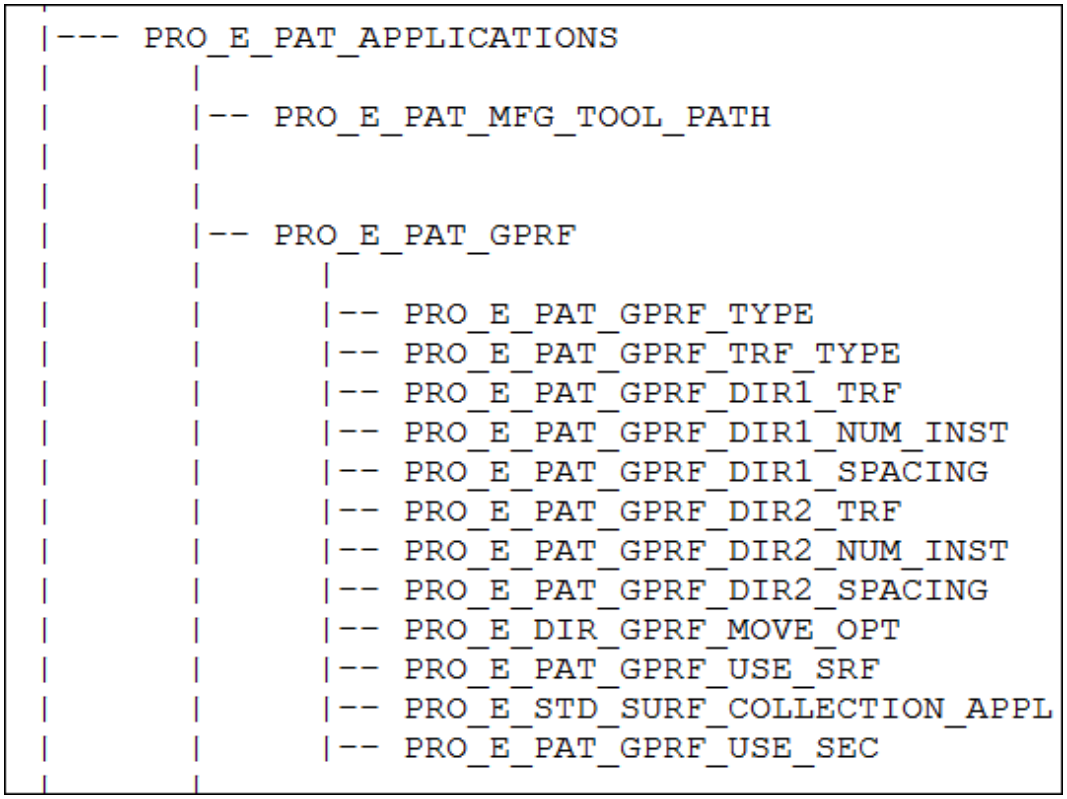
For more information on the patterning options related to an NC sequence, refer to Manufacturing section of the Creo Parametric help.

Geometry Pattern Recognition

You can read the Pattern Recognition feature parameters using the element PRO_E_PAT_GPRF described in this section. You cannot create the Pattern Recognition feature in Creo Parametric TOOLKIT using these elements.

The following figure shows the element tree for Geometry Pattern Recognition application:

Pattern Element Tree for Geometry Pattern Recognition



The pattern recognition compound element `PRO_E_PAT_GPRF` contains the following elements:

- `PRO_E_PAT_GPRF_TYPE`—Specifies the type of geometry pattern to be recognized: **Identical** or **Similar**. It takes the integer values: `0` for **Identical** and `1` for **Similar**.
- `PRO_E_PAT_GPRF_TRF_TYPE`—Specifies the recognized geometry patterns. It takes the integer value: `0` for **Direction**, `1` for **Axis** and `2` for **Spatial**.
- `PRO_E_PAT_GPRF_DIR1_TRF`—Specifies the first direction of transformation. It takes the following integer values from the enumerated type `ProGenPatternDirectionType`:
  - `PRO_GENPAT_TRANSLATIONAL`—Specifies `-1` for translational pattern.
  - `PRO_GENPAT_DIR1_ROTATIONAL`—Specifies `58` for first direction for rotational pattern.

- PRO_E_PAT_GPRF_DIR1_NUM_INST—Specifies the number of members in the first direction or in the angular direction.
- PRO_E_PAT_GPRF_DIR1_SPACING—Specifies the spacing between members in the first direction or the angle between members in the angular direction.
- PRO_E_PAT_GPRF_DIR2_TRF—Specifies the second direction of transformation. It takes the following integer values from the enumerated type ProGenPatternDirectionType:
  - PRO_GENPAT_TRANSLATIONAL—Specifies -1 for translational pattern.
  - PRO_GENPAT_DIR2_ROTATIONAL—Specifies 60 for second direction for rotational pattern.
- PRO_E_PAT_GPRF_DIR2_NUM_INST—Specifies the number of members in the second direction or in the angular direction.
- PRO_E_PAT_GPRF_DIR2_SPACING—Specifies the spacing between members in the second direction or the angle between members in the angular direction.
- PRO_E_DIR_GPRF_MOVE_OPT—Specifies if a pattern of copy-move features must be created by the geometry pattern recognition feature. Specify 1 if you want the number of pattern members and spacing to be modified.
- PRO_E_PAT_GPRF_USE_SRF—Specifies 1 if the members in the geometry pattern recoginition feature have been limited with surfaces. This element restricts the pattern recognition to a limited region on the model.
- PRO_E_STD_SURF_COLLECTION_APPL—Specifies the collection of surfaces that define the leader of the geometry pattern to be recognized.
- PRO_E_PAT_GPRF_USE_SEC—Specifies 1 if a sketch has been used to limit the members in the geometry pattern recognition feature.

The following table lists the contents of each PRO_E_PAT_GPRF element.

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_PAT_GPRF_TYPE	Type of recognized pattern	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_GPRF_TRF_TYPE	Type of transformation	PRO_VALUE_TYPE_INT	0, 1 or 2
PRO_E_PAT_GPRF_DIR1_TRF	First transformation direction	PRO_VALUE_TYPE_INT	—1, 58 or 60
PRO_E_PAT_GPRF_DIR1_NUM_INST	Number of instances in the first direction	PRO_VALUE_TYPE_INT	
PRO_E_PAT_GPRF_DIR1_SPACING	Spacing in the first direction	PRO_VALUE_TYPE_DOUBLE	
PRO_E_PAT_GPRF_DIR2_TRF	Second transformation direction	PRO_VALUE_TYPE_INT	—1, 58 or 60

Element ID Values	Element Name	Data Type	Valid Values
PRO_E_PAT_GPRF_DIR2_NUM_INST	Number of instances in the second direction	PRO_VALUE_TYPE_INT	
PRO_E_PAT_GPRF_DIR2_SPACING	Spacing in the second direction	PRO_VALUE_TYPE_DOUBLE	
PRO_E_DIR_GPRF_MOVE_OPT	Create pattern of move features	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_PAT_GPRF_USE_SRF	Use surface for region definition	PRO_VALUE_TYPE_INT	0 or 1
PRO_E_STD_SURF_COLLECTION_APPL	Reference surfaces	PRO_VALUE_TYPE_SELECTION	
PRO_E_PAT_GPRF_USE_SEC	Sketch for region definition	PRO_VALUE_TYPE_INT	0 or 1

For more information on the geometry pattern recognition, refer to the Creo Parametric help.

Obtaining the Element Tree for a Pattern

Function Introduced:

- **ProPatternElemtreeCreate()**

To obtain the element tree for a pattern, call the function `ProPatternElemtreeCreate()`. You can then use the element tree read-access functions described in the sections Feature Elements on page 774 (on page 29 - 13) and Feature Element Paths on page 772 (on page 29 - 11), such as the functions `ProElement*Get()`, `ProElement*Visit()`, and `ProElementArrayGet()`).

Note

Inspection of Fill, Axis, and Directional patterns is not supported via the element tree in Pro/ENGINEER Wildfire 2.0. `ProPatternElemtreeCreate()` returns `PRO_TK_NOT_IMPLEMENTED` for fill patterns.

Visiting and Creating a Pattern

Functions Introduced:

- **ProPatternMemberVisit()**
- **ProPatternMembersGet()**

- **ProPatternCreate()**
- **ProPatternInAssemblyCreate()**

The function `ProPatternMemberVisit()` visits the feature members in a pattern. This function takes the visit action function `ProFeatureVisitAction()` and the filter action function `ProFeatureFilterAction()` as its input arguments. The function `ProFeatureFilterAction()` is a generic action function for filtering features from a pattern. It returns the filter status of the features in the pattern. This status is used as an input argument by the function `ProFeatureVisitAction()`.

The function `ProPatternMembersGet()` returns the feature members in a pattern. For a group pattern, the output argument is a group pattern feature.

When your element tree is complete, create the pattern by calling the function `ProPatternCreate()`. This function requires as input the feature (`ProFeature`) to be patterned and the pattern class (feature or group) of the new pattern.

To obtain the `ProPattern` handle for the new pattern, call the function `ProFeaturePatternGet()` with the same input feature as `ProPatternCreate()`. For more information on the function `ProFeaturePatternGet()`, refer to the section Manipulating Patterns on page 144 in the Core: Features on page 131 chapter.

The function `ProPatternInAssemblyCreate()` creates a pattern in the assembly that is provided in the element tree. The input parameters are as follows:

- *p_component_path*—The component path specified using the structure `ProAsmcomppath`.
- *pattern_feature*—Feature defined by the `ProFeature` object.
- *pat_class*—Pattern class defined by the enumerated data type `ProPatternClass`.
- *elem_tree*—The root element of the pattern element tree.

You must specify the pattern object for this function because a feature pattern can be a part of both a group pattern and a feature pattern.

The function returns the error `PRO_TK_ABORT` if the pattern feature creation failed. You must ensure that you use a new pattern name every time you create a new pattern.

43

Element Trees: Sections

A section is a parametric two-dimensional cross section used to define the shape of three-dimensional features, such as extrusions. In Creo Parametric, you create a section interactively using Sketcher mode. In a Creo Parametric TOOLKIT application, you can create sections completely programmatically using the functions described in this section.

Overview

A section is a parametric two-dimensional model used to define the shape of three-dimensional features in parts and assemblies. When using Creo Parametric interactively, you create a section using Sketcher mode. In a Creo Parametric TOOLKIT application, you can create sections completely programmatically using the functions described in this section.

Sections fall into two types: 2D and 3D. Both types are represented by the object `ProSection` (an opaque handle) and manipulated by the same functions.

The difference between the types arises out of the context in which the section is being used, and affects the requirements for the contents of the section and also of the feature element tree in which it is placed when creating a sketched feature.

Put simply, a 2D section is self-contained, whereas a 3D section contains references to 3D geometry in a parent part or assembly.

You can use Intent Datums such Intent Axis, Intent Point, Intent Plane, and Intent Coordinate System as references for sketcher dimensions. You can use Intent Point and Intent Axis to create sections using projections.

In a Creo Parametric TOOLKIT application, you can work with a section either in an Intent Manager or a non-Intent Manager mode. In the non-Intent Manager mode, if you make any changes to the section, you must solve and regenerate the section to apply the changes. On the other hand, in the Intent Manager mode, all the changes are applied immediately.

You can create section constraints programmatically using the Intent Manager property. This corresponds to creating sections within the Intent Manager mode in Creo Parametric.

This chapter is concerned with 2D sections, which are the simplest. The extra steps required to construct a 3D section are described in the chapter Element Trees: Sketched Features on page 1004, which follows this one.

Creating Section Models

A 2D section, because it is self-contained, may be stored as a Creo Parametric model file. It then has the extension `.sec`.

The steps required to create and save a section model using Creo Parametric TOOLKIT follow closely those used in creating a section interactively using Sketcher mode in Creo Parametric.

To Create and Save a Section Model

1. Allocate the two-dimensional section and define its name.
2. Add section entities (lines, arcs, splines, and so on) to define the section geometry, in section coordinates.
3. Add section dimensions that parametrically drive the shape of the entities.
4. Solve and regenerate the section.
5. Save the section.

When you are creating a section that is to be used in a sketched feature, Steps 1 and 5 will be replaced by different techniques. these techniques are described fully in the chapter on Element Trees: Sketched Features on page 1004.

The steps are described in more detail in the following sections.

Allocating a Two-Dimensional Section

Functions Introduced:

- **ProSection2DAlloc()**
- **ProSectionFree()**
- **ProSectionNameSet()**
- **ProSectionNameGet()**

A two-dimensional section is identified in Creo Parametric TOOLKIT by an opaque pointer called `ProSection`. This type, and the functions in this section, are declared in the include file `ProSection.h`.

The function `ProSection2DAlloc()` allocates memory for a new, standalone section and outputs a `ProSection` handle to identify it. All the other Creo Parametric TOOLKIT functions that operate on sections take this `ProSection` as their first input argument.

The function `ProSectionNameSet()` enables you to set the name of a section. Calling this function places the section in the Creo Parametric namelist and enables it to be recognized by Creo Parametric as a section model in the database.

The following code fragment shows how to use these two functions.

```
ProSection    section;
ProName       wname;


ProSection2DAlloc (&section);
ProStringToWstring (wname, "demo");
ProSectionNameSet (section, wname);
```

Such sections created programmatically are in the non-Intent Manager mode by default.

To free a section allocated with `ProSection2DAlloc()`, you must use `ProSectionFree()`.

Setting the Mode of a Section

Functions Introduced:

- **ProSectionIntentManagerModeGet()**
- **ProSectionIntentManagerModeSet()**

Use the function `ProSectionIntentManagerModeGet()` to check if the Intent Manager property is ON or OFF for the specified section.

Use the function `ProSectionIntentManagerModeSet()` to set the Intent Manager property to ON or OFF for the specified section. This function must be called before using the other Creo Parametric TOOLKIT functions to access sections with the Intent Manager property set to ON.

Copying the Current Section

Functions Introduced:

- **ProSectionActiveGet()**
- **ProSectionActiveSet()**

Use the function `ProSectionActiveGet()` to create a copy of the section that you are using currently; this copy is created within the same Sketcher session. The mode of such a section depends on the current Sketcher mode. Starting from Pro/ENGINEER Wildfire 5.0, the Intent Manager mode is the default sketcher mode. Use the function `ProSectionFree()` to free the memory allocated to the section obtained with the function `ProSectionActiveGet()`.

Use the function `ProSectionActiveSet()` to set the specified section as the current active Sketcher section.

Note

The function call `ProSectionActiveSet()` makes the **Undo** and **Redo** menu options available in Creo Parametric.

Section Constraints

Functions Introduced:

- **ProSectionConstraintsIdsGet()**
- **ProSectionConstraintsGet()**
- **ProSectionConstraintDeny()**
- **ProSectionConstraintCreate()**
- **ProSectionConstraintDelete()**

The function `ProSectionConstraintsIdsGet()` returns an array of section constraint identifiers that currently exist in the specified section.

Note

You must solve the section first by calling the function `ProSectionSolve()` to get the section constraints. Because adding or deleting section entities might invalidate the current list of section constraint identifiers, you must solve the section again to get the up-to-date list.

If a section has not been fully dimensioned with dimensions created explicitly by the user, the Sketcher will make assumptions in order to solve the section. If the Sketcher can assume enough constraints to find a unique solution to the section, it solves the section successfully.

However, you might want to disable certain Sketcher constraints to have more control over the way the section is dimensioned and solved. To do this, use the function `ProSectionConstraintDeny()` to deny a certain section constraint.

Note

The function `ProSectionConstraintDeny()` is not supported for sections that have the Intent Manager property set to ON.

The function `ProSectionConstraintsGet()` returns information about the specified section constraint. It takes as input the section handle and the constraint identifier for which the information is requested. The function returns details about the section constraint including its type, status, and references. The constraint types are defined in the include file `ProSecConstr.h`. The following table lists the possible constraint types.

Constraint Type	Description
`PRO_CONSTRAINT_SAME_POINT`	Make the points coincident.
`PRO_CONSTRAINT_HORIZONTAL_ENT`	Make the entity horizontal.
`PRO_CONSTRAINT_VERTICAL_ENT`	Make the entity vertical.
`PRO_CONSTRAINT_PNT_ON_ENT`	Place the point on the entity.

Constraint Type	Description
PRO_CONSTRAINT_TANGENT_ENTS	Make the entities tangent.
PRO_CONSTRAINT_ORTHOG_ENTS	Make the entities perpendicular.
PRO_CONSTRAINT_EQUAL_RADII	Make the arcs or circles of equal radius.
PRO_CONSTRAINT_PARALLEL_ENTS	Make the entities parallel.
PRO_CONSTRAINT_EQUAL_SEGMENTS	Make the segments of equal length.
PRO_CONSTRAINT_COLLINEAR_LINES	Make lines co-linear.
PRO_CONSTRAINT_90_ARC	Make the arcs 90 degrees.
PRO_CONSTRAINT_180_ARC	Make the arcs 180 degrees.
PRO_CONSTRAINT_HORIZONTAL_ARC	Make the arcs horizontal.
PRO_CONSTRAINT_VERTICAL_ARC	Make the arcs vertical.
PRO_CONSTRAINT_SYMMETRY	Impose symmetry.
PRO_CONSTRAINT_SAME_COORD	Assume the endpoints and centers of arcs to have the same coordinates.

The possible types of constraint status are as follows:

- PRO_TK_CONSTRAINT_DENIED—The constraint is denied. This gives you more control over the section.
- PRO_TK_CONSTRAINT_ENABLED—The constraint is enabled. The Sketcher uses the predefined assumption.

Use the function ProSectionConstraintCreate() to create constraints between entities in the specified section. Use the function ProSectionConstraintDelete() to delete the specified section constraint.

Note

The function ProSectionConstrainCreate() works only if the Intent Manager property of the specified section is set to ON.

Solving and Regenerating a Section

Functions Introduced:

- **ProSectionEpsilonGet()**
- **ProSectionEpsilonSet()**
- **ProSectionSolve()**
- **ProSectionSolveRigid()**
- **ProSecdimValueGet()**

- **ProSecdimValueSet()**
- **ProSectionRegenerate()**

Although the action of the **Regenerate** command in Sketcher mode is seen as a single operation by the Creo Parametric user, it is in fact composed of two distinct actions. These two operations are invoked separately from a Creo Parametric TOOLKIT application. The two operations are as follows:

- Solving—Calculating the way in which the geometry of the entities is driven by the dimensions. It is at this stage that Sketcher constraints are applied, under- or over-dimensioning is discovered and reported, and values are assigned to new dimensions.
- Regenerating—Reconstructing the geometry of the section to obey the current dimension values.

You invoke these stages using the functions `ProSectionSolve()` and `ProSectionRegenerate()`, respectively.

Note

The `ProSectionSolve()` and `ProSectionRegenerate()` are not supported for sections that have the Intent Manager property set to ON.

You must solve a programmatically-created section before using it to build three-dimensional geometry. You need to regenerate the section only if you have explicitly modified the dimension values since you solved the section.

When you create a section interactively using Sketcher mode, you normally adjust the values of dimensions after the first regeneration, because the initial values assigned to them correspond to the free-hand, initial sketch and are therefore not exact. When you create a section with Creo Parametric TOOLKIT , the entities are usually created with exactly the geometry needed in the finished section. Therefore, although solving is always necessary, it is not usually necessary to explicitly reset dimension values or regenerate the section.

Solving a section in Creo Parametric TOOLKIT involves applying the same constraints used in interactive Sketcher mode. Creo Parametric TOOLKIT , like the Sketcher, identifies situations of near symmetry in the section, assumes them to be intended as exact symmetry, and constrains them to be symmetrical in future regenerations. For example, lines that are nearly the same length are assumed to be intended to be the same length, and are therefore constrained to be so.

The function `ProSectionSolveRigid()` solves the specified section by fixing the coordinates of all the section entities with respect to a coordinate system. In this way, the section entities do not have to be solved individually. To use this function, a coordinate system within the section must exist; the function uses the first coordinate system found in the section.

Note

You must ensure that the added section entities are correct because potential errors will not be solved and may show up only during later stages.

When there are a lot of section entities, this function dramatically reduces the amount of time required to solve a section.

Note

The function `ProSectionSolveRigid()` is not supported for sections that have the Intent Manager property set to ON.

Epsilon is the tolerance value, which is used to set the proximity for automatic finding of constraints. Use the function `ProSectionEpsilonSet()` to set the value for epsilon. For example, if your section has two lines that differ in length by 0.5, set the epsilon to a value less than 0.5 to ensure that `ProSectionSolve()` does not constrain the lines to be the same length. To get the current epsilon value for the section, use the function `ProSectionEpsilonGet()`.

Please note the following important points related to epsilon:

- Epsilon determines the smallest possible entity in a section. If an entity is smaller than epsilon, then the entity is considered to be a degenerate entity. Degenerate entity is an entity which cannot be solved. It causes solving and regenerating of the section to fail. For example, a circle with radius 0 or line with length 0 are considered as degenerate entities.
- There are many types of constraints, and epsilon has a different meaning for each type. For example, consider two points. For the constraint `PRO_CONSTRAINT_SAME_POINT`, epsilon is the minimum distance between the two points beyond which the points will be treated as separate points. If the distance between the two points is within the epsilon value, the two points are treated as coincident points.
- Creo Parametric has a default value set for epsilon. This value is also used in the Sketcher user interface.
- If the input geometry is accurate and the user does not want the solver to change it by adding constraints, then set the value of epsilon to 1E-9.
- If the input geometry is nearly accurate and the user wants the solver to guess

the intent by adding constraints and further aligning the geometry, then in this case epsilon should reflect the maximal proximity between geometry to be constrained.

- You cannot set the value of epsilon to zero.

The functions `ProSecdimValueGet()` and `ProSecdimValueSet()` enable you to access the value of a dimension. If you change dimension values, you must call `ProSectionRegenerate()` to recalculate the new section shape.

Automatic Section Dimensioning

Function Introduced:

- **ProSectionAutodim()**

The function `ProSectionAutodim()` is used to automatically add needed dimensions to a section to make it fully constrained. It takes as input a `ProSection` handle and a pointer to the opaque structure called `ProWSecerror`. Before calling this function, be sure to allocate the pointer to `ProWSecerror` using `ProSecerrorAlloc()`. Any errors resulting from the call to the function `ProSectionAutodim()` are stored in the `ProWSecerror` structure. To free the allocated memory, call the function `ProSecerrorFree()`.

The `ProSectionAutodim()` function can be used on a section where no dimensions have been created yet, as well as on a partially dimensioned section.

If dimensions have been added successfully, the function `ProSectionAutodim()` also solves the input section.

Note

The function `ProSectionAutodim()` is not supported for sections that have the Intent Manager property set to ON.

Adding Section Entities

Functions Introduced:

- **ProSectionEntityAdd()**
- **ProSectionEntityDelete()**
- **ProSectionEntityReplace()**

The function ProSectionEntityAdd() takes as input the ProSection that identifies the section, and a pointer to a user-visible structure called Pro2dEntdef, which defines the entity.

The Pro2dEntdef structure is a generic structure that contains only a field indicating the type of entity. For each type of entity, there is a dedicated structure that has the entity type as its first field; these structures are named Pro2dLinedef, Pro2dArcdef, and so on. The Creo Parametric TOOLKIT application builds up the structure appropriate to the entity to be added, and inputs it to ProSectionEntityAdd() by casting its address to (Pro2dEntdef*). The entity structures are declared in the include file Pro2dEntdef.h.

The function ProSectionEntityAdd() outputs an integer that is the identifier of the new entity within the section. The Creo Parametric TOOLKIT application needs these values because they are used to refer to entities when adding dimensions.

The following code fragment demonstrates how to add a single line entity.

```
Pro2dLinedef   line;
int            line_id;

line.type    = PRO_2D_LINE;
line.end1[0] = 0.0;
line.end1[1] = 0.0;
line.end2[0] = 10.0;
line.end2[1] = 0.0;

ProSectionEntityAdd (section,
    (Pro2dEntdef*)&line, &line_id);
```

The function ProSectionEntityDelete() enables you to delete a section entity from the specified section.

The function ProSectionEntityReplace() enables you to replace an existing entity from the specified section with another entity in the same section. This functionality enables you to redefine an existing section programmatically.

To use the function ProSectionEntityReplace(), you must first add the new entity to the section (to get its identifier), then replace the old entity identifier with the new one.

Accessing Selection Reference of the Entity

Functions Introduced:

- **ProSectionEntityGetSelected()**

The function ProSectionEntityGetSelected() provides the references of the selected entity. The input arguments of this function are:

- *handle*—The section handle.
- *entity_id*—The identifier of the section entity.
- *pnt_type*—Specifies the type of point selection on the entity. The valid values are:
  - `PRO_ENT_WHOLE`—Specifies the whole entity.
  - `PRO_ENT_START`—Specifies the start point of the entity.
  - `PRO_ENT_END`—Specifies the end point of the entity.
  - `PRO_ENT_CENTER`—Specifies the center of the entity.
  - `PRO_ENT_LEFT_TANGENT`—Specifies the point on the entity, where the normalized param value is 0.5.
  - `PRO_ENT_RIGHT_TANGENT`—Specifies the point on the entity, where the normalized param value is 0.0.
  - `PRO_ENT_TOP_TANGENT`—Specifies the point on the entity, where the normalized param value is 0.25.
  - `PRO_ENT_BOTTOM_TANGENT`—Specifies the point on the entity, where the normalized param value is 0.75.
- *pnt*—Specifies the location of the point on the entity geometry.
- *idx_pnt*—Specifies the index of interpolation point relative to *pnt_type*, if the section entity is a spline. In this case the value of the argument *pnt* is ignored. Specify `PRO_VALUE_UNUSED` if the entity is not a spline.

This function creates the selection object programmatically for use in functions that require selection references of entities as the input.

Construction Entities

Functions Introduced:

- **ProSectionEntityIsConstruction()**
- **ProSectionEntityConstructionSet()**

Use the function `ProSectionEntityIsConstruction()` to determine if the specified entity is a construction entity. Construction entities are used for reference and are not used to create feature geometry.

The function `ProSectionEntityConstructionSet()` sets the specified entity to be of type construction.

Modifying Entities

Functions Introduced:

- **ProSectionEntityIntersectionGet()**
- **ProSectionEntityParamEval()**
- **ProSectionEntityCorner()**
- **ProSectionEntityDivide()**

The function `ProSectionEntityIntersectionGet()` returns the intersection points between the two section entities. Use the function `ProArrayFree()` to free the memory.

The function `ProSectionEntityParamEval()` to find the corresponding normalized parameter on the curve, given the XY point.

The function `ProSectionEntityCorner()` trims the selected entities to each other. The selected entities may not intersect with each other. The entities may be trimmed by either cutting them or extending them.

Use the function `ProSectionEntityDivide()` to divide a section entity into two or more new entities. If the entity is dimensioned then delete the dimensions before dividing it.

Adding Section Dimensions

Functions Introduced:

- **ProSecdimCreate()**
- **ProSecdimDelete()**
- **ProSecdimDiameterSet()**
- **ProSecdimDiameterClear()**
- **ProSecdimDiameterInquire()**
- **ProSecdimStrengthen()**
- **ProSecdimLockSet()**
- **ProSecdimIsLocked()**

When you create a dimension interactively in Sketcher mode, you select entities and points on entities and Creo Parametric deduces from those picks what type of dimension is being added. When you add a dimension using the function `ProSecdimCreate()`, you must specify the dimension type. The dimension types are defined in the include file `ProSecdimTypes.h`. The following table lists the possible values.

Constant	Description
`PRO_TK_DIM_LINE`	Length of a line
`PRO_TK_DIM_LINE_POINT`	Distance between a line and a vertex
`PRO_TK_DIM_RAD`	Radius of an arc or a circle
`PRO_TK_DIM_DIA`	Diameter of an arc or a circle

Constant	Description
PRO_TK_DIM_LINE_LINE	Distance between two lines
PRO_TK_DIM_PNT_PNT	Distance between two points
PRO_TK_DIM_PNT_PNT_HORIZ	Distance between two points (X coordinates)
PRO_TK_DIM_PNT_PNT_VERT	Distance between two points (Y coordinates)
PRO_TK_DIM_AOC_AOC_TAN_HORIZ	Horizontal distance between two arcs or circles
PRO_TK_DIM_AOC_AOC_TAN_VERT	Vertical distance between two arcs or circles
PRO_TK_DIM_ARC_ANGLE	Angle of an arc
PRO_TK_DIM_LINES_ANGLE	Angle between two lines
PRO_TK_DIM_LINE_AOC	Distance between a line and an arc or a circle
PRO_TK_DIM_LINE_CURVE_ANGLE	Angle between a spline and a line
PRO_TK_DIM_3_PNT_ANGLE	Angular dimension defined by three points
PRO_TK_DIM_DIA_LINEAR	Linear diameter dimension
PRO_TK_DIM_PNT_PNT_ORI	Distance between two points in specified orientation
PRO_TK_DIM_AOC_AOC_ORI	Distance between two arcs or circles in specified orientation
PRO_TK_DIM_TOT_INC_ANG	Total included angle
PRO_TK_DIM_ANG_POLAR	Angle between the x-axis and a vector. The vector is defined by two points

The function ProSecdimCreate() takes several input arguments, including the following:

- int *entity_ids[]*—An array of integers that are the identifiers of the section entities to which the dimension refers.
- ProSectionPointType *point_types[]*—A dimension can reference a vertex (the end of an entity), the center of an arc or a circle, a line or circle itself (the whole entity), or tangent points on an arc or a circle. To specify these types of dimension reference points, specify the appropriate point type constant for each dimension in the entity_ids array. These constants are listed in the include file ProSecdimType.h.
- int *num_ids*—The number of section dimension identifiers in the entity_ids array. This is typically 1 or 2 (line length versus a point-to-point dimension).
- ProSecdimType *dim_type*—The type of section dimension to create, as listed in the ProSecdimType.h file.
- Pro2dPnt *place_pnt*—The two-dimensional location of the dimension label. This is equivalent to the middle mouse button pick when you are using Sketcher mode.

  Note that the position of this label can sometimes determine the exact role of the dimension. For example, a dimension of type PRO_TK_DIM_LINES_ANGLE may refer to the acute or obtuse angle between two lines, depending on where the label is positioned.

The ProSecdimCreate() function outputs the identifier of the dimension, which is needed to identify the dimension if its value needs to be changed at a later time.

Note

The dimensions do not need to be given values to create a complete and correct section of any form. See the section Solving and Regenerating a Section on page 992 for a detailed explanation of the assignment of values.

The following code fragment shows how to create a dimension for the length of a line entity.

```
int                    line_id[1], width_dim;
Pro2dPnt               point;
ProSectionPointType    pnt_type[1];

line_id[0] = 1;
point[0] = 5.0;
point[1] = 1.0;
pnt_type[0] = PRO_ENT_WHOLE;

ProSecdimCreate (section, line_id, pnt_type, 1,
                 PRO_TK_DIM_LINE, point,
                 &width_dim);
```

The following code fragment shows how to create a dimension for the horizontal distance between two arc ends.

```
int                    arc1_id, arc2_id, arc1_end2, arc2_end1,
                       dist_dim;
Pro2dPnt               point;
int                    entities[2];
ProSectionPointType    pnt_types[2];

pnt_types[0] = PRO_ENT_START;
pnt_types[1] = PRO_ENT_END;
entity[0]    = arc1_end2;
entity[1]    = arc2_end1;
point[0]     = 5.0;
point[1]     = 5.0;

ProSecdimCreate (section, entities, pnt_types, 2,
                 PRO_TK_DIM_PNT_PNT_HORIZ, point, &dist_dim);
```

The ProSecdimDiam...() functions extend the dimension creation functionality to include diameters for sections used to create revolved features. Function ProSecdimDiamSet() converts a specified section dimension (between a centerline and another entity) into a diameter dimension.

ProSecdimDiamClear() does the opposite, converting a diameter dimension into a regular one. Use function ProSecdimDiamInquire() to determine if a dimension is a diameter dimension.

The function ProSecdimStrengthen() converts a weak dimension to a strong dimension.

You can lock or unlock sketch dimensions. Locking of dimensions avoids modifications to the sections outside the sketcher mode. The function ProSecdimIsLocked() determines whether a sketch dimension is locked. Use the function ProSecdimLockSet() to lock or unlock a specified dimension.

Example 1: Creating Spline Point Dimensions in Sections

The sample code in Ug3DSectSplineDim.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a Section dimension between the first and last interpolation point of a Spline.

Error Reporting

Functions Introduced:

- **ProSecerrorAlloc()**
- **ProSecerrorCount()**
- **ProSecerrorMsgGet()**
- **ProSecerrorItemGet()**
- **ProSecerrorAdd()**
- **ProSecerrorFree()**

Both ProSectionSolve() and ProSectionRegenerate() might result in a list of errors about the entities in the section. These errors are stored in an opaque structure called ProWSecerror. Before calling one of these functions, use ProSecerrorAlloc() to allocate memory for an error structure, then pass the pointer to the error structure to ProSectionSolve() or ProSectionRegenerate().

You can add application-specific section errors to an error structure. To do this, call the function ProSecerrorAdd().

The function ProSecerrorCount() tells you how many error messages are contained in the error structure. The errors themselves are identified by sequential integers, so you can step through the list. Use the function ProSecerrorMsgGet() to get the text of each message. Use the function

ProSecerrorItemGet() to get the identifier of the problem entity that caused a specific error message. To free the allocated memory, call the function ProSecerrorFree().

A Creo Parametric TOOLKIT application that builds sections generally aims to make them complete and correct without any interactive help from the Creo Parametric user. Therefore, the errors reported by the functions ProSectionSolve() and ProSectionRegenerate() are directed at the Creo Parametric TOOLKIT developer as a debugging aid, rather than at the final Creo Parametric user.

The following code fragment shows a call to ProSectionSolve() and an analysis of the errors produced.

```
ProWSecerror   errors;
int            n_errors, e;
ProError       status;
ProMsg         wmsg;
char           msg[PRO_PATH_SIZE];
int            ent_id;

ProSecerrorAlloc (&errors);
status = ProSectionSolve (section, &errors);
if (status != PRO_TK_NO_ERROR)
{
    ProSecerrorCount (&errors, &n_errors);
    for (e = 0; e < n_errors; e++)
    {
         ProSecerrorMsgGet (errors, e, wmsg);
         ProWstringToString (msg, wmsg);
         ProSecerrorItemGet (errors, e, &ent_id);
         printf ("%s: Problem ID, %d\n", msg, ent_id);
    }
    ProSecerrorFree (&errors);
    return (-1);
}
```

Retrieving and Saving a Section

Functions Introduced:

- **ProFeatureNumSectionsGet()**
- **ProFeatureSectionCopy()**

To retrieve a section from disk, use the function ProMdlnameRetrieve() with the model type PRO_2DSECTION. You can save a section to a file using the function ProMdlSave().

You can also retrieve or copy a section from a feature. The function ProFeatureNumSectionsGet() finds the number of sections in the specified feature. Given a feature handle and section index,

`ProFeatureSectionCopy()` initializes and returns a section handle to a section copied from the specified feature. Memory for this section is controlled by the Creo Parametric TOOLKIT application and must therefore be freed by a call to `ProSectionFree()`.

Example 1: Creating a Section Model

The sample code in `UgSectModelCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` illustrates how to use all the functions described in this chapter to create a section model.

44

Element Trees: Sketched Features

This chapter describes the Creo Parametric TOOLKIT functions that enable you to create and manipulate sketched features.

Sketched features are features that require one or more sections to completely define the feature, such as extruded and revolved protrusions.

This chapter outlines the necessary steps to programmatically create sketched features using Creo Parametric TOOLKIT.

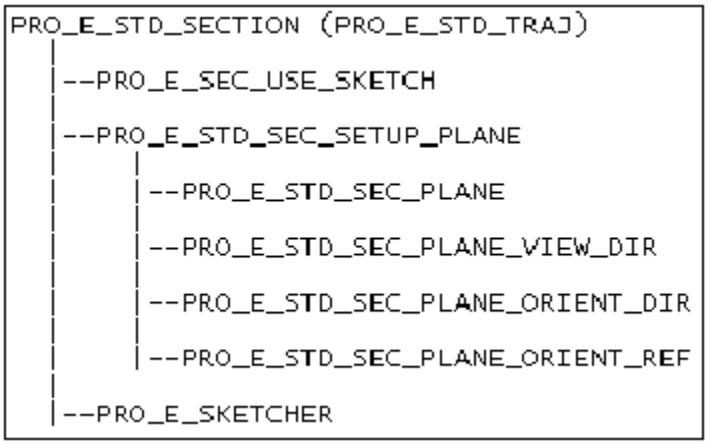
Overview

The chapter Element Trees: Principles of Feature Creation on page 764 explains how to create a simple feature using the feature element tree, and the documentation in the chapter on Element Trees: Sections on page 987 explains how to create a section. This chapter explains how to put these methods together, with a few additional techniques, to create features that contain sketched sections.

Element Tree for Sketched Features

The element tree of any feature that contains a sketch contains a subtree that identifies the sketch object and describes how it is positioned in the model. As this subtree is the same for every sketched feature, it is documented in its own header file, called `ProStdSection.h`. The diagram below shows the structure of that subtree.

Element Tree for Sketched Features



The subtree of the `PRO_E_STD_SEC_SETUP_PLANE` element defines the sketch plane, the location of the sketch plane, the orientation plane and the orientation direction, and the viewing direction. You can use Intent Planes as sketch orientation or placement references.

The element `PRO_E_SKETCHER` is of type `POINTER`, and its value is the object `ProSection`, introduced in the documentation in the chapter on Sections.

The element `PRO_E_SEC_USE_SKETCH` refers to any valid selected 'Sketch' (sketched datum curve) suitable to the current Sketch Based Feature. When this element is used, the sketch will be stored in the feature as a reference to the sketch feature, and no internal section, sketch plane or orientation will be required. Using this element in the `PRO_E_STD_SECTION` element tree allows a feature to be created in one step, without creating the feature first as incomplete.

Note

The use of internal sections, and the process by which an internal-section based feature is created, remains unchanged in Pro/ENGINEER Wildfire 2.0.

The following table shows the sketched features that are supported by Creo Parametric TOOLKIT, the names of the corresponding header files which show their element trees, and the IDs of the elements in each tree which contain the standard sketch subtree as shown in the above figure Element Tree for Sketched Features

Feature	Header	Element containing Subtree
Extrude	ProExtrude.h	PRO_E_STD_SECTION
Revolve	ProRevolve.h	PRO_E_STD_SECTION
Rib	ProRib.h	PRO_E_STD_SECTION
Hole	ProHole.h	PRO_E_SKETCHER (2D) PRO_E_STD_SECTION
Fill (Flat datum surface)	ProFlatSrf.h	PRO_E_STD_SECTION
Draft	ProDraft.h	PRO_E_STD_SECTION
Sketched datum curve	ProDtmCrv.h	PRO_E_STD_SECTION
Sketched datum point	ProDtmCrv.h	PRO_E_STD_SECTION
Simple (constant) sweep	ProSweep.h	PRO_E_SWEEP_SPINE PRO_E_SWEEP_SECTION(2D)

Creating Features Containing Sections

The chapter Element Trees: Principles of Feature Creation on page 764 explained that to create a feature from an element tree, you build the tree of elements using `ProElementAlloc()`, `ProElemtreeElementAdd()`, and so on, and then call `ProFeatureCreate()` to create the feature using the tree. If the feature is to contain a sketch, the sequence is a little more complex.

As explained in the documentation in the section chapter on Element Trees: Sections on page 987, a 2D section stored in a model file can be allocated by calling `ProSection2dAlloc()`. Instead, Creo Parametric must allocate as part of the initial creation of the sketched feature, a section that will be part of a feature. The allocation is done by calling `ProFeatureCreate()` with an element tree which describes at minimum the feature type and form, in order to create an incomplete feature. In creating the feature, Creo Parametric calculates the location and orientation of the section, and allocates the `ProSection` object. This section is then retrieved from the value of the `PRO_E_SKETCHER` element that is found in the element tree extracted from the created feature. Fill the empty section using `ProSection` related functions.

After adding the section contents and the remaining elements in the tree, add the new information to the feature using `ProFeatureRedefine()`.

To Create Sketched Features Element Trees

1. Build an element tree but do not include the element `PRO_E_SKETCHER`.
2. Call `ProFeatureCreate()` with the option `PRO_FEAT_CR_INCOMPLETE_FEAT`, so that the incomplete element tree is accepted.
3. Extract the value of the element `PRO_E_SKETCHER` created by Creo Parametric from an element tree extracted using `ProFeatureElemtreeExtract()` on the incomplete feature.
4. Using that value as the `ProSection` object, create the necessary section entities and dimensions, and solve the section.
5. Add any other elements not previously added to the tree, such as extrusion depth. The depth elements may also be added before the creation of incomplete feature (before step 2).
6. Call `ProFeatureRedefine()` with the completed element tree.

Creating Features with 2D Sections

Sketched features using 2D sections do not require references to other geometry in the Creo Parametric model. Some examples of where 2D sections are used are:

- Base features, sometimes called first features. This type of feature must be the first feature created in the model, and be of type `PRO_FEAT_FIRST_FEAT`.
- Sketched hole features.
- The `PRO_E_SWEEP_SECTION` section of a simple sweep feature.

To create 2D sketched features, follow the steps outlined in the section To Create Sketched Features Element Trees on page 1007.

Note

For 2D sketched features, you need not specify section references or use projected 3D entities. Entities in a 2D section are dimensioned to themselves only. A 2D section does not require any elements in the tree to setup the sketch plane or the orientation of the sketch. Thus, the `PRO_E_STD_SEC_SETUP_PLANE` subtree is not included.

Verifying Section Shapes

Function Introduced:

- **ProSectionShapeGet()**

Certain features may or may not be able to use a section due to the shape of the section. Different sketched feature tools such as extrude and revolve have different requirements for sections. For example, solid protrusions contain only closed and non-intersecting sections. Solid cuts or datum surfaces have open non-intersecting sections. Fill features must have closed sections.

After the section is regenerated, `ProSectionShapeGet()` obtains the shape of a given section. This information is used to determine if the section is acceptable for feature creation.

The section shapes are as follows:

Constant	Function	Description
`PRO_SECSHAPE_EMPTY`	`ProSectionShapeGet()`	An empty section
`PRO_SECSHAPE_POINTS`	`ProSectionShapeGet()`	Section contains only sketched datum points
`PRO_SECSHAPE_1_OPEN_LOOP`	`ProSectionShapeGet()`	Section contains a single open loop (and possibly points)
`PRO_SECSHAPE_1_CLOSED_LOOP`	`ProSectionShapeGet()`	Section contains a single closed loop (and possibly points)
`PRO_SECSHAPE_MIXED_LOOPS`	`ProSectionShapeGet()`	Section contains at least one open and one closed loop (and possibly points)
`PRO_SECSHAPE_MULTI_OPEN_LOOPS`	`ProSectionShapeGet()`	Section contains multiple open loops (and possibly points)
`PRO_SECSHAPE_MULTI_CLOSED_LOOPS`	`ProSectionShapeGet()`	Section contains multiple closed loops (and possibly points)
`PRO_SECSHAPE_INTERSECTING`	`ProSectionShapeGet()`	Section contains loops that intersect each other (and possibly points)

Note

Use geometry entities and not construction entities to define section shapes that are then used to create solid or surface geometry. To convert the construction entities to geometry entities, use the function `ProSectionEntityConstructionSet()` with the input argument *construction* set to `PRO_B_FALSE`.

Creating Features with 3D Sections

A 3D section needs to define its location with respect to the existing geometrical features. The subtree contained in the element `PRO_STD_SEC_SETUP_PLANE` defines the location of the sketch planeEdge entities; any other 2D entities in the sketch must be dimensioned to those entities, so that their 3D location is fully defined.

3D Section Location in the Owning Model

Function Introduced:

- **ProSectionLocationGet()**

For a 2D section in a feature, Creo Parametric decides where the section will be positioned in 3D.

If the section is 3D, the feature tree elements below `PRO_E_STD_SEC_SETUP_PLANE` specify the sketch plane, the direction from which it is being viewed, an orientation reference, and a direction which that reference represents (TOP, BOTTOM, LEFT or RIGHT). When you call `ProFeatureCreate()`, this information is used to calculate the 3D plane in which the section lies, and its orientation in that plane.

The position of the section origin in the plane is not implied by the element tree, and cannot be specified by the Creo Parametric TOOLKIT application: position is chosen arbitrarily by Creo Parametric. This is because the interactive user of Creo Parametric never deals in absolute coordinates, and doesn't need to specify, or even know, the location of the origin of the section. In Creo Parametric TOOLKIT describe all section entities in terms of their coordinate values, so you need to find out where Creo Parametric has put the origin of the section. This is the role of the function `ProSectionLocationGet()`.

`ProSectionLocationGet()` provides the transformation matrix that goes from 2D coordinates within the section to 3D coordinates of the owning part or assembly. This is equivalent to describing the position and orientation of the 2D section coordinate system with respect to the base coordinate system of the 3D model.

So `ProSectionLocationGet()` can be called in order to calculate where to position new section entities so that they are in the correct 3D position in the part or assembly.

Reference Entities and Use Edge

Functions introduced:

- **ProSectionEntityFromProjection()**
- **ProSectionEntityIsProjection()**

- **ProSectionEntityUseEdge()**
- **ProSectionEntityUseEdgeLoop()**
- **ProSectionEntityUseEdgeChain()**
- **ProSectionEntityReferenceGet()**

The previous section explained how to set the correct 3D position of new section entities. You also need to make the entities parametric, that is, to ensure that Creo Parametric knows how to calculate their new positions during regeneration.

When sketching a section using Creo Parametric, entities are positioned parametrically by dimensioning them or aligning them to items in the 3D model. Creo Parametric TOOLKIT does not allow you to explicitly align section entities, but you can add dimensions which relate section entities to 3D entities in the owning model. You can do this using references. A reference entity represents a position in the section of an item in a 3D model that is used as a dimension reference. The reference entity itself does not give rise to 3D geometry in the owning feature. Reference entities are visible in interactive sketcher operations; they are shown as dashed and are used during autodimensioning and alignment operations.

In Creo Parametric TOOLKIT reference entities are created using `ProSectionEntityFromProjection()`. This function takes as input a `ProSelection` describing the 3D model entity being projected, and outputs the integer ID of the resulting known section entity. This ID is used to specify the attachment of a section dimension, as described in the documentation in the section chapter Element Trees: Sections on page 987. Reference entities are included in the output of `ProSectionEntityIdsGet()`, but can be distinguished from regular section entities by calling the function `ProSectionEntityIsProjection()`.

To align a section entity with a 3D model entity, project the 3D entity to create a reference entity, and then either add a dimension between this reference entity and the one to be aligned or use `ProSectionAutodim()` to do this.

To create a regular section entity whose geometry is itself an exact projection of a 3D model entity, create it and align it in a single step using the function `ProSectionEntityUseEdge()`. This function has the same arguments as `ProSectionEntityFromProjection()`, and it creates a reference entity in the same way, but it requires an additional step of copying the reference entity to a regular entity with the same geometry. It outputs the ID of the regular entity it creates. The ID of the reference entity is always 1 less than the ID of the regular entity.

`ProSectionEntityUseEdge()` is equivalent to the Creo Parametric sketcher command **Use Edge**. The functions `ProSectionEntityUseEdgeLoop()` and `ProSectionEntityUseEdgeChain()` allow you to execute a **Use Edge** operation on multiple edges simultaneously.

Note

If you create the known and projected entities first, you need not call `ProSectionLocationGet()` as described above; instead you can look at the geometry of the known and projected entities, and then position the new entities relative to the projected entities.

The function `ProSectionEntityReferenceGet()` provides 3D geometry that is a reference for a projected entity in a given section.

Creating Geometry by Offsetting

The functions described in this section enable you to create offset entities from edges and 3D curve segments from normal, chain and loop selection.

Functions Introduced:

- **ProSectionEntityUseOffset()**
- **ProSectionEntityUseOffsetChain()**
- **ProSectionEntityUseOffsetLoop()**

The function `ProSectionEntityUseOffset()` creates a sketched entity that is offset at a specified distance from a single edge. This function takes as input a `ProSelection` object describing the 3D model entity.

The function `ProSectionEntityUseOffsetChain()` creates sketched entities that are offset from a chain of edges or entities and the function `ProSectionEntityUseOffsetLoop()` creates sketched entities offset from a loop of edges or entities.

The behavior of the functions in this section is similar to the offset operation achieved using **Sketch ► Offset** in Creo Parametric.

Reusing Existing Sketches

Functions introduced:

- **ProFeatureSketchAdd()**
- **ProFeatureSketchedWithOptionsCreate()**

Creo Parametric allows you to copy sections from previously created features into new sketched features.

The function `ProFeatureSketchAdd()` copies the selected section from one feature to another feature.

The function `ProFeatureSketchedCreate()` has been deprecated. Use the function `ProFeatureSketchedWithOptionsCreate()` instead. The function `ProFeatureSketchedWithOptionsCreate()` creates a feature from the element tree, and also copies the sketched sections to the new feature. This reduces the sketched feature creation effort to a single Creo Parametric TOOLKIT function call. The element tree must contain all of the required elements except the `PRO_E_STD_SECTION` subtree.

Example 1: Creating an Extruded Protrusion Base Feature

The sample code in `UgSktExtrusionCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows the creation of an extruded protrusion as a base feature.

Example 2: Creating a Sketched Datum Curve

The sample code in `UgSketchedCurveCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows the creation of a sketched datum curve using the conventional approach.

45

Element Trees: Extrude and Revolve

This chapter describes how to use the include files `ProExtrude.h`, and `ProRevolve.h` so that you can create extruded and revolved features programmatically. As Extrude and Revolve features are sketched features; we recommend you to read the chapters Element Trees: Principles of Feature Creation on page 764 and Element Trees: Sketched Features on page 1004 before referring to this chapter.

The Element Tree for Extruded Features

The element tree for extrude features is documented in the header file `ProExtrude.h`. The functions `ProFeatureTypeGet()` and `ProFeatureSubtypeGet()` return an Extrude feature. The types of Extrude features are:

- Protrusion
- Cut
- Surface
- Surface Trim
- Thin Protrusion
- Thin Cut
- Sheetmetal Cut
- Sheetmetal Unattached Wall

Refer to the chapter Production Applications: Sheetmetal on page 1310 for element details on sheetmetal features.

The extrude element tree contains toggles to switch between different feature types. An extruded feature tree also contains subtrees supporting the section and depth parameters for the feature.

You can use Intent Datums such as Intent Point, Intent Axis, and Intent Plane for depth reference in extrude features.

The following figure shows the element tree for extruded features.

The Element Tree for Extruded Feature

```
PRO_E_FEATURE_TREE
  |
  |--PRO_E_STD_FEATURE_NAME
  |
  |--PRO_E_EXT_SURF_CUT_SOLID_TYPE
  |
  |--PRO_E_REMOVE_MATERIAL
  |
  |--PRO_E_BODY
  |
  |--PRO_E_IS_SMT_CUT
  |
  |--PRO_E_SMT_CUT_NORMAL_DIR
  |
  |--PRO_E_FEATURE_FORM
  |
  |--PRO_E_STD_SECTION
  |
  |--PRO_E_FEAT_FORM_IS_THIN
  |
  |--PRO_E_STD_MATRLSIDE
  |
  |--PRO_E_THICKNESS
  |
  |--PRO_E_SRF_END_ATTRIBUTES
  |
  |--PRO_E_TRIM_QLT_SIDE
  |
  |--PRO_E_TRIM_QUILT
  |
  |--PRO_E_STD_DIRECTION
  |
  |--PRO_E_STD_EXT_DEPTH
  |
  |--PRO_E_STD_SMT_THICKNESS
  |
  |--PRO_E_STD_SMT_SWAP_DRV_SIDE
  |
  |--PRO_E_SMT_WALL_SHARPS_TO_BENDS
  |
  |--PRO_E_SMT_PUNCH_AXIS_PNT
  |
  |--PRO_E_SMT_FILLETS
  |
  |--PRO_E_SMT_DEV_LEN_CALCULATION
  |
  |--PRO_E_SMT_PUNCH_TOOL_DATA
  |
  |--PRO_E_SMT_MERGE_DATA
  |
  |--PRO_E_EXT_COMP_DRFT_ANG
  |
  |--PRO_E_FEAT_THIN
```

The elements are assigned values depending on the type of extrusion you want to create.

The following table lists the common elements for all types of extrusions and their permissible values:

Element ID	Value
PRO_E_FEATURE_TYPE	Feature type, not required for creation: PRO_FEAT_PROTRUSION PRO_FEAT_CUT PRO_FEAT_DATUM_SURFN
PRO_E_FEATURE_FORM	Mandatory = PRO_EXTRUDE
PRO_E_EXT_SURF_CUT_SOLID_TYPE	Mandatory Of type ProExtFeatType = PRO_EXT_FEAT_TYPE_SOLID for Solid feature type = PRO_EXT_FEAT_TYPE_SURFACE for Surface feature type
PRO_E_REMOVE_MATERIAL	Material Removal Of type ProExtRemMaterial = PRO_EXT_MATERIAL_ADD for a Protruded feature = PRO_EXT_MATERIAL_REMOVE for a Cut feature
PRO_E_STD_SECTION	Standard section elements
PRO_E_BODY	Compound Element. Specifies the body options.
PRO_E_STD_DIRECTION*	Direction of creation. Of type ProExtDirection = PRO_EXT_CR_IN_SIDE_ONE for depth in side one = PRO_EXT_CR_IN_SIDE_TWO for depth in side two
PRO_E_STD_MATRLSIDE*	Direction of material affected with respect to the sketch. Required for all cuts, all thin features, and for solid protrusions with open sections.
PRO_E_STD_EXT_DEPTH	Compound Element. Specifies the depth type and value for the extrude feature.
PRO_E_EXT_DEPTH_TO	Compound Element. Specifies the depth type and value for Side 1, that is, extrusion in the first direction from the sketch plane.

Element ID	Value
PRO_E_EXT_DEPTH_TO_TYPE	Mandatory element. Specifies the type of depth for Side 1. The depth type is specified using the enumerated data type ProExtDepthToType. The valid values are: • PRO_EXT_DEPTH_TO_BLIND—Extrudes a section from the sketching plane to the specified depth value. • PRO_EXT_DEPTH_TO_NEXT—Extrudes a section from the sketching plane to the first surface that it reaches. • PRO_EXT_DEPTH_TO_ALL—Extrudes a section from the sketching plane to the last surface it reaches. • PRO_EXT_DEPTH_TO_UNTIL—Extrudes a section to intersect with a selected surface. • PRO_EXT_DEPTH_TO_REF—Extrudes a section to a selected point, curve, plane, or surface. • PRO_EXT_DEPTH_SYMMETRIC—Extrudes a section on each side of the sketching plane by half of the specified depth value in each direction.
PRO_E_EXT_DEPTH_TO_REF	Specifies the reference element for Side 1, when the depth type is PRO_EXT_DEPTH_TO_REF or PRO_EXT_DEPTH_TO_UNTIL. The valid reference types are: • PRO_SURFACE • PRO_AXIS • PRO_EDGE • PRO_CURVE • PRO_POINT • PRO_EDGE_START • PRO_EDGE_END • PRO_CRV_START • PRO_CRV_END • PRO_BODY
PRO_E_EXT_DEPTH_TO_REF_TRF	Specifies the options available for the depth type PRO_EXT_DEPTH_TO_REF for Side 1. The depth type is specified using the enumerated data type ProExtDepthRefOpt. The valid values are: • PRO_EXT_DEPTH_REF_NONE—Extrudes a section to a selected point, curve, plane, or surface. • PRO_EXT_DEPTH_REF_OFFS—Extrudes a section to an offset of the selected point, curve, plane, or surface. • PRO_EXT_DEPTH_REF_TRNSLT—Extrudes a section to a translation of the selected point, curve, plane, or surface.

Element ID	Value
PRO_E_EXT_DEPTH_TO_REF_TRF_VAL	Specifies the offset or translation value for Side 1, when the depth type is PRO_EXT_DEPTH_TO_REF, and the option type is PRO_EXT_DEPTH_REF_OFFS or PRO_EXT_DEPTH_REF_TRNSLT.
PRO_E_EXT_DEPTH_TO_VALUE	Specifies the value of depth for Side 1, when the depth type is PRO_EXT_DEPTH_TO_BLIND or PRO_EXT_DEPTH_SYMMETRIC.
PRO_E_EXT_DEPTH_FROM	Compound Element. Specifies the depth type and value for Side 2, that is, extrusion in the second direction from the sketch plane.
PRO_E_EXT_DEPTH_FROM_TYPE	Mandatory element. Specifies the type of depth for Side 2. The depth type is specified using the enumerated data type ProExtDepthFromType. The valid values are: • PRO_EXT_DEPTH_FROM_BLIND—Extrudes a section from the sketching plane to the specified depth value. • PRO_EXT_DEPTH_FROM_NEXT—Extrudes a section from the sketching plane to the first surface that it reaches. • PRO_EXT_DEPTH_FROM_ALL—Extrudes a section from the sketching plane to the last surface it reaches. • PRO_EXT_DEPTH_FROM_UNTIL—Extrudes a section to intersect with a selected surface. • PRO_EXT_DEPTH_FROM_REF—Extrudes a section to a selected point, curve, plane, or surface. • PRO_EXT_DEPTH_FROM_NONE—Extrudes a section only on Side 1 from the sketch plane, no extrusion on Side 2.
PRO_E_EXT_DEPTH_FROM_REF	Specifies the reference element for Side 2, when the depth type is PRO_EXT_DEPTH_FROM_REF or PRO_EXT_DEPTH_FROM_UNTIL. The valid reference types are: • PRO_SURFACE • PRO_AXIS • PRO_EDGE • PRO_CURVE • PRO_POINT • PRO_EDGE_START • PRO_EDGE_END • PRO_CRV_START • PRO_CRV_END • PRO_BODY
PRO_E_EXT_DEPTH_FROM_REF_TRF	Specifies the options available for the depth type PRO_EXT_DEPTH_FROM_REF for Side 2. The depth type is specified using the enumerated data

Element ID	Value
	type ProExtDepthRefOpt. The valid values are: • PRO_EXT_DEPTH_REF_NONE—Extrudes a section to a selected point, curve, plane, or surface. • PRO_EXT_DEPTH_REF_OFFS—Extrudes a section to an offset of the selected point, curve, plane, or surface. • PRO_EXT_DEPTH_REF_TRNSLT—Extrudes a section to a translation of the selected point, curve, plane, or surface.
PRO_E_EXT_DEPTH_FROM_REF_TRF_VAL	Specifies the offset or translation value for Side 2, when the depth type is PRO_EXT_DEPTH_FROM_REF, and the option type is PRO_EXT_DEPTH_REF_OFFS or PRO_EXT_DEPTH_REF_TRNSLT.
PRO_E_EXT_DEPTH_FROM_VALUE	Specifies the value of depth for Side 2, when the depth type is PRO_EXT_DEPTH_FROM_BLIND.
PRO_E_STD_FEATURE_NAME	Default given by application depending on the feature type. Can be modified by the user.
PRO_E_EXT_COMP_DRFT_ANG	Draft Compound Element that allows you to add a draft on the extrude feature.
PRO_E_EXT_DRFT_ANG	Draft of type ProExtDrftAng. • PRO_EXT_DRFT_ANG_NO_DRAFT—To create extruded features without a draft. • PRO_EXT_DRFT_ANG_DRAFT—To create extruded features with a draft.
PRO_E_EXT_DRFT_ANG_VAL	The draft angle. The draft angle can have value between [-89.9, 89.9].
PRO_E_FEAT_THIN	Compound element. It specifies how to close a thin feature when one or more surfaces can be used to cap, that is, close the feature and attach it to the solid geometry. Here the sketch is an open sketch.
PRO_E_FEAT_THIN_STRT	Compound element. It specifies the options for the first end point of the thin feature.
PRO_E_FEAT_THIN_STRT_OPT	Specifies how to cap the first end point of the thin feature using the enumerated data type ProFeatThinOpt. The valid values are: • PRO_FEAT_THIN_IGNORE—Caps the feature as a free end. When you specify this value, the feature is created with a free end even if a reference edge or surface is available to cap the feature. • PRO_FEAT_THIN_DEFAULT—Caps the feature to the specified edge or surface.
PRO_E_FEAT_THIN_STRT_REF	Specifies the edge or surface that must be used to cap and attach the first end point of the feature to the solid geometry.
PRO_E_FEAT_THIN_END	Compound element. It specifies the options for the second end point of the feature.

Element ID	Value
PRO_E_FEAT_THIN_END_OPT	Specifies how to cap the second end point of the thin feature using the enumerated data type ProFeatThinOpt. The valid values are: • PRO_FEAT_THIN_IGNORE—Caps the feature as a free end. When you specify this value, the feature is created with a free end even if a reference edge or surface is available to cap the feature. • PRO_FEAT_THIN_DEFAULT—Caps the feature to the specified edge or surface.
PRO_E_FEAT_THIN_END_REF	Specifies the edge or surface that must be used to cap and attach the second end point to the solid geometry.

Elements identified with '*' depend on the definition of the standard section. These elements are not assigned values until the standard section has been completely allocated (which typically happens during redefine of the feature). Values assigned to these elements while the section is not complete are ignored.

The following table lists the elements needed to create extruded features, in addition to those already discussed:

Feature Type	Element ID	Comment
Solid	PRO_E_EXT_COMP_DRFT_ANG	Compound element to specify draft options.
	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Mandatory. Specifies the body to add geometry to. The valid values are: • PRO_BODY_USE_NEW—The geometry in the feature is stored in the new body. • PRO_BODY_USE_SELECTED—The geometry in the feature is stored in the single selected body.
	PRO_E_BODY_SELECTED	Specifies the reference to the selected body. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED **Note** Only single reference is allowed.
Thin	PRO_E_THICKNESS	Mandatory >= 0.0

Feature Type	Element ID	Comment
		Of type PRO_VALUE_TYPE_DOUBLE
	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Mandatory. The valid values for PRO_BODY_USE_NEW and PRO_BODY_USE_SELECTED are same as Solid.
	PRO_E_BODY_SELECTED	Same as Solid.
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProExtFeatForm = PRO_EXT_FEAT_FORM_NO_THIN for a feature not having Thin = PRO_EXT_FEAT_FORM_THIN for a Thin feature
	PRO_E_STD_MATRLSIDE	Mandatory Of type ProExtMatlSide
	PRO_E_FEAT_THIN	Compound element to specify options to cap and attach thin features to solid geometry.
Solid Cut	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Mandatory. Specifies the body features that cuts the geometry. The valid values are: • PRO_BODY_USE_ALL—The geometry in the feature is cut by all the existing bodies. • PRO_BODY_USE_SELECTED—The geometry in the feature is stored in the selected bodies.
	PRO_E_BODY_SELECTED	Specifies the reference to the selected bodies. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED **Note** Multiple references are allowed.
	PRO_E_STD_MATRLSIDE	Mandatory Of type ProExtMatlSide
	PRO_E_EXT_COMP_DRFT_ANG	Draft compound element for features that do not have feature

Feature Type	Element ID	Comment
		form as Thin.
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProExtFeatForm = PRO_EXT_FEAT_FORM_NO_THIN for a feature not having Thin = PRO_EXT_FEAT_FORM_THIN for a Thin feature
Thin Cut	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Same as Solid Cut
	PRO_E_BODY_SELECTED	Same as Solid Cut
	PRO_E_STD_MATRLSIDE	Mandatory Of type ProExtMatlSide
	PRO_E_THICKNESS	Mandatory >= 0.0 Of type PRO_VALUE_TYPE_DOUBLE
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProExtFeatForm = PRO_EXT_FEAT_FORM_NO_THIN for a feature not having Thin = PRO_EXT_FEAT_FORM_THIN for a Thin feature
Surface	PRO_E_SRF_END_ATTRIBUTES	Mandatory Of type ProExtSurfEndAttr It must be assigned at the same time or after the section is fully completed.
	PRO_E_EXT_COMP_DRFT_ANG	Draft compound element for features that do not have feature form as Thin.
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProExtFeatForm = PRO_EXT_FEAT_FORM_NO_THIN for a feature not having Thin = PRO_EXT_FEAT_FORM_THIN for a Thin feature
Surface Trim	PRO_E_STD_MATRLSIDE	Mandatory Of type ProExtMatlSide
	PRO_E_TRIM_QUILT	Mandatory Of type Quilt
	PRO_E_TRIM_QLT_SIDE	Mandatory

Feature Type	Element ID	Comment
		Of type`ProExtTrimQltSide`
	`PRO_E_FEAT_FORM_IS_THIN`	Of Type ProExtFeatForm = `PRO_EXT_FEAT_FORM_NO_THIN` for a feature not having Thin = `PRO_EXT_FEAT_FORM_THIN` for a Thin feature
	`PRO_E_EXT_COMP_DRFT_ANG`	Draft compound element for features that do not have feature form as Thin.
Thin Surface Trim	`PRO_E_STD_MATRLSIDE`	Mandatory Of type `ProExtMatlSide`
	`PRO_E_THICKNESS`	Mandatory >= 0.0 Of type `PRO_VALUE_TYPE_DOUBLE`
	`PRO_E_TRIM_QUILT`	Mandatory Of type Quilt
	`PRO_E_TRIM_QLT_SIDE`	Mandatory Of type `ProExtTrimQltSide` if `PRO_E_STD_MATRLSIDE` is “both”. Must be assigned at the same time as `PRO_E_STD_MATRLSIDE`.
	`PRO_E_FEAT_FORM_IS_THIN`	Of Type ProExtFeatForm = `PRO_EXT_FEAT_FORM_NO_THIN` for a feature not having Thin = `PRO_EXT_FEAT_FORM_THIN` for a Thin feature

Examples: Creating Extruded Features

The following examples demonstrate creation of extrude features of various forms. These examples are adapted from an example template file `UgSktExtrusionTemplate.c` available on the Creo Parametric load point under `protoolkit/protk_appls/pt_userguide/ptu_featcreat`.

- Example 1: Creating an Extruded Feature on page 1024
- Example 2: To Create an Extruded Cut with Two-sided Thru-all Depth on page 1024

Conventional Approach

Example 1: Creating an Extruded Feature

The sample code in the file `UserSktExtrusionProtrusion.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an Extruded Protrusion by the conventional approach for sketched features. The example creates an incomplete feature using `ProFeatureCreate()`, extracts the section from the element tree of the incomplete feature, builds the section on the section handle obtained, and, completes the feature using `ProFeatureRedefine()`.

The user is prompted to select the sketching and the orientation planes and then the reference edges for the sketch. The user is also required to enter the X and Y offsets to be applied to the sketch from the projected edges.

Example 2: To Create an Extruded Cut with Two-sided Thru-all Depth

The sample code in the file `UgSktExtrusionCut.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an extruded cut with two-sided thru-all depth.

Example 3: To Create an Extruded Thin Cut

The sample code in the file `UgSktExtrusionCut.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an extruded thin cut. Its depth is two-sided, up to a selected reference.

Example 4: To Create an Extruded Datum Surface Feature

The sample code in the file `UgSktExtrusionSurfaceCapped.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an extruded datum surface feature. Its depth is one-side blind.

Example 5: To Create a Surface Trim Extruded Feature

The sample code in the file `UgSktExtrusionSurfaceTrim.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an extruded surface trim. Its depth is one-side blind.

Direct Creation Approach

The sample code in the file `UgSktExtrusionCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create an extruded protrusion by the direct approach for sketched features introduced in Pro/ENGINEER Wildfire. The user is prompted to select a sketched datum curve feature that is used as a section for the created protrusion.

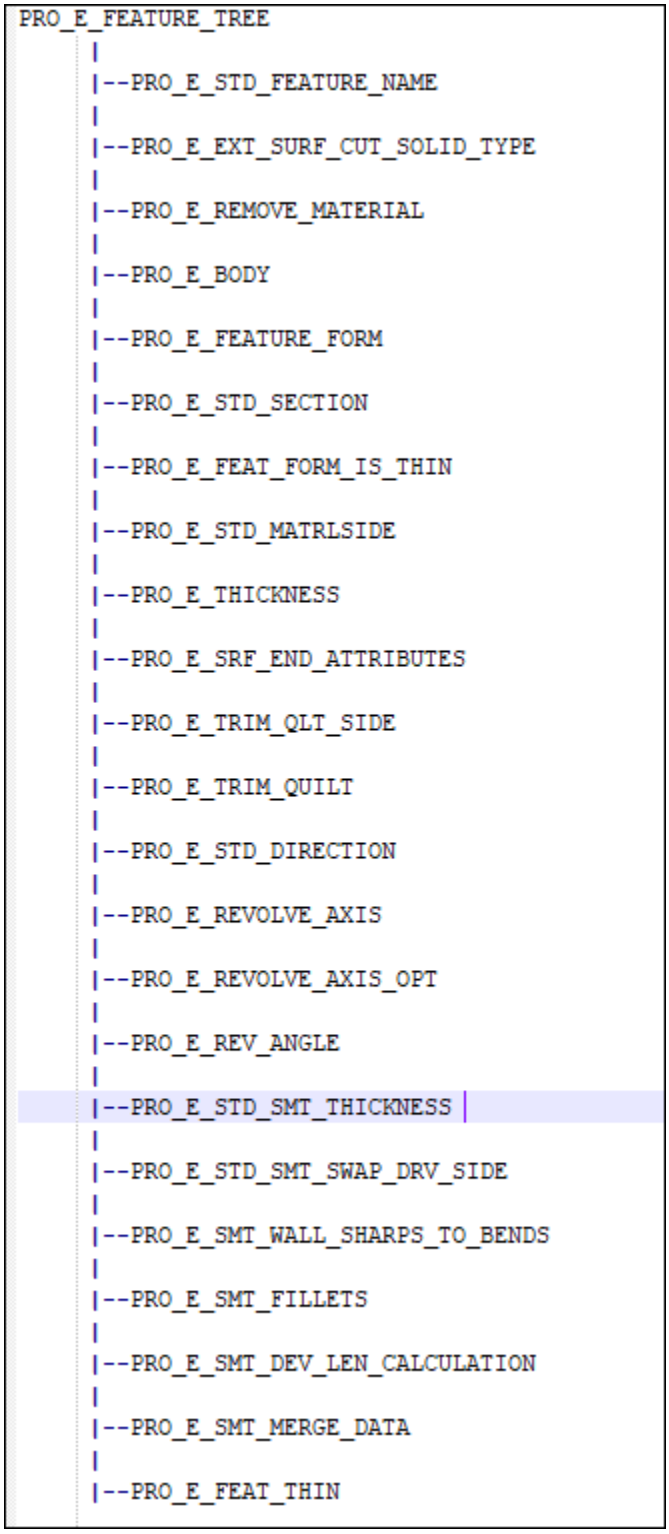
The Element Tree for Revolved Features

The element tree for revolved features is documented in the header file `ProRevolve.h`, and has a fairly simple structure. It shows that, apart from the usual elements for the tree root and feature name, a revolved feature tree contains the elements to make the feature a solid protrusion, a thin protrusion, a solid cut, a thin cut, a surface, a surface trimmed feature, or a thin surface trimmed feature.

You can use Intent Datums such as Intent Axis, Intent Plane, and Intent Point for placement references.

The following figure shows the element tree for revolved features.

Element Tree for Revolved Feature



The elements are assigned values depending on the type of revolved feature you want to create.

The following table lists the common elements for all types of revolved features and their permissible values:

Element ID	Value
PRO_E_FEATURE_TYPE	Feature type: PRO_FEAT_PROTRUSION PRO_FEAT_CUT PRO_FEAT_DATUM_SURF Not required for creation.
PRO_E_FEATURE_FORM	Mandatory= PRO_REVOLVE
PRO_E_EXT_SURF_CUT_SOLID_TYPE	Mandatory Of type ProRevFeatType = PRO_REV_FEAT_TYPE_SOLID for Solid feature type = PRO_REV_FEAT_TYPE_SURFACE for Surface feature type
PRO_E_FEAT_FORM_IS_THIN	Feature Form Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for a Solid feature = PRO_REV_FEAT_FORM_THIN for a Thin feature
PRO_E_REMOVE_MATERIAL	Material Removal Of type ProRevRemMaterial = PRO_REV_MATERIAL_ADD for a Protruded feature = PRO_REV_MATERIAL_REMOVE for a Cut feature
PRO_E_STD_SECTION	Standard section elements.
PRO_E_BODY	Compound Element
PRO_E_STD_DIRECTION*	Direction of creation. Of type ProRevDirection = PRO_REV_CR_IN_SIDE_ONE

Element ID	Value
	for angle in side one = PRO_REV_CR_IN_SIDE_TWO for angle in side two
PRO_E_STD_MATRLSIDE*	Direction of material affected with respect to the sketch. It is required for all cuts, all thin features, and for solid protrusions with open sections.
PRO_E_REVOLVE_AXIS_OPT	Optional, of the type ProRevAxisOptAttr. Identifies if the axis to revolve about is a part of the sketch or an external datum axis.
PRO_E_REVOLVE_AXIS	Optional. Reference to external datum axis, if PRO_E_REVOLVE_AXIS = PRO_REV_AXIS_EXT_REF.
PRO_E_REV_ANGLE	Compound Element
PRO_E_REV_ANGLE_TO	Compound Element
PRO_E_REV_ANGLE_TO_TYPE	Mandatory Of type ProRevAngleToType
PRO_E_REV_ANGLE_TO_VAL	Depends on PRO_E_REV_ANGLE_TO_TYPE Of type PRO_VALUE_TYPE_DOUBLE (in degrees)
PRO_E_REV_ANGLE_TO_REF	Depends on PRO_E_REV_ANGLE_TO_TYPE Of type listed in the Angle Type table that follows.
PRO_E_REV_ANGLE_FROM	Compound Element
PRO_E_REV_ANGLE_FROM_TYPE	Mandatory Of type ProRevAngleFromType
PRO_E_REV_ANGLE_FROM_VAL	Depends on PRO_E_REV_ANGLE_FROM_TYPE Of type PRO_VALUE_TYPE_DOUBLE (in degrees)
PRO_E_REV_ANGLE_FROM_REF	Depends on PRO_E_REV_ANGLE_FROM_TYPE Of type listed in the Angle Type table that follows.
PRO_E_STD_FEATURE_NAME	Default given by application depending on the feature type. Can be modified by the user.
PRO_E_FEAT_THIN	Compound element. It specifies how to close a thin feature when one or more surfaces can be used to cap, that is, close the feature and attach it to solid geometry. Here the sketch is an open sketch.
PRO_E_FEAT_THIN_STRT	Compound element. It specifies the options for the first end point of the thin feature.
PRO_E_FEAT_THIN_STRT_OPT	Specifies how to cap the first end point of the thin feature using the enumerated data type ProFeatThinOpt. The valid values are:

Element ID	Value
	• PRO_FEAT_THIN_IGNORE—Caps the feature as a free end. When you specify this value, the feature is created with a free end even if a reference edge or surface is available to cap the feature. • PRO_FEAT_THIN_DEFAULT—Caps the feature to the specified edge or surface.
PRO_E_FEAT_THIN_STRT_REF	Specifies the edge or surface that must be used to cap and attach the first end point to the solid geometry.
PRO_E_FEAT_THIN_END	Compound element. It specifies the options for the second end point of the thin feature.
PRO_E_FEAT_THIN_END_OPT	Specifies how to cap the second end point of the thin feature using the enumerated data type ProFeatThinOpt. The valid values are: • PRO_FEAT_THIN_IGNORE—Caps the feature as a free end. When you specify this value, the feature is created with a free end even if a reference edge or surface is available to cap the feature. • PRO_FEAT_THIN_DEFAULT—Caps the feature to the specified edge or surface.
PRO_E_FEAT_THIN_END_REF	Specifies the edge or surface that must be used to cap and attach the second end point to the solid geometry.

Elements identified with '*' depend on the definition of the standard section. These elements may not be assigned values until the standard section has been completely allocated (which typically happens during redefine of the feature). Values assigned to these elements while the section is not complete are ignored.

The following table lists the angle types for revolved features along with possible valid references:

Angle Type	Valid Reference Types
PRO_REV_ANGLE_TO_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END, PRO_SURFACE (Plane).
PRO_REV_ANGLE_FROM_REF	PRO_POINT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END, PRO_SURFACE (Plane).

The following table lists the elements needed to create revolved features, in addition to those already discussed:

Feature Type	Element ID	Comment
Solid	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Mandatory. Specifies the body to add geometry to. The valid values are:

Feature Type	Element ID	Comment
		• PRO_BODY_USE_NEW—The geometry in the feature is stored in the new body. • PRO_BODY_USE_SELECTED—The geometry in the feature is stored in the single selected body.
	PRO_E_BODY_SELECTED	Specifies the reference to the selected body. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED **Note** Only single reference is allowed.
Thin	PRO_E_STD_MATRLSIDE	Mandatory Of type ProRevMatlSide
	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Same as Solid
	PRO_E_BODY_SELECTED	Same as Solid
	PRO_E_THICKNESS	Mandatory >= 0.0 Of type PRO_VALUE_TYPE_DOUBLE
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for feature not having Thin = PRO_REV_FEAT_FORM_THIN for a Thin feature
	PRO_E_FEAT_THIN	Compound element to specify options to cap and attach thin features to solid geometry.
Solid Cut	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Mandatory. Specifies the body features that cuts the geometry. The valid values are:

Feature Type	Element ID	Comment
		• PRO_BODY_USE_ALL—The geometry in the feature is cut by all the existing bodies. • PRO_BODY_USE_SELECTED—The geometry in the feature is stored in the selected bodies.
	PRO_E_BODY_SELECTED	Specifies the reference to the selected bodies. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED **Note** Multiple references are allowed.
	PRO_E_STD_MATRLSIDE	Mandatory Of type ProRevMatlSide
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for feature not having Thin = PRO_REV_FEAT_FORM_THIN for a Thin feature
Thin Cut	PRO_E_BODY	Compound element
	PRO_E_BODY_USE	Same as Solid Cut
	PRO_E_BODY_SELECTED	Same as Solid Cut
	PRO_E_STD_MATRLSIDE	Mandatory Of type ProRevMatlSide
	PRO_E_THICKNESS	Mandatory >= 0.0 Of type PRO_VALUE_TYPE_DOUBLE

Feature Type	Element ID	Comment
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for feature not having Thin = PRO_REV_FEAT_FORM_THIN for a Thin feature
Surface	PRO_E_SRF_END_ATTRIBUTES	Mandatory Of type ProRevSurfEndAttr Must be assigned at the same time or after the section is fully completed.
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for feature not having Thin = PRO_REV_FEAT_FORM_THIN for a Thin feature
Surface Trim	PRO_E_STD_MATRLSIDE	Mandatory Of type ProRevMatlSide
	PRO_E_TRIM_QUILT	Mandatory Of type Quilt
	PRO_E_TRIM_QLT_SIDE	Mandatory Of type ProRevTrimQltSide if PRO_E_STD_MATRLSIDE is "both". Must be assigned at the same time as PRO_E_STD_MATRLSIDE.
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for feature not having Thin = PRO_REV_FEAT_FORM_THIN for a Thin feature
Thin Surface Trim	PRO_E_STD_MATRLSIDE	Mandatory Of type ProRevMatlSide
	PRO_E_THICKNESS	Mandatory >= 0.0 Of type PRO_VALUE_TYPE_DOUBLE
	PRO_E_TRIM_QUILT	Mandatory Of type Quilt

Feature Type	Element ID	Comment
	PRO_E_TRIM_QLT_SIDE	Mandatory Of type ProRevTrimQltSide
	PRO_E_FEAT_FORM_IS_THIN	Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for feature not having Thin = PRO_REV_FEAT_FORM_THIN for a Thin feature

Examples: Creating Revolved Features

The following examples demonstrate creation of revolved features of various forms. These examples are adapted from an example template file UgSktRevolveTemplate.c available on the Creo Parametric loadpoint under <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat.

- Example 6: To Create a Revolved Protrusion on page 1033
- Example 7: To Create a Revolved Thin Cut on page 1033
- Example 8: To Create a Revolved Surface on page 1033

Example 6: To Create a Revolved Protrusion

The sample code in the file UgSktRevolveProtrusion.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a revolved protrusion feature with symmetric depth.

Example 7: To Create a Revolved Thin Cut

The sample code in the file UserSktRevolveThinCut.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a revolved thin cut, with independent angular dimensions for both sides.

Example 8: To Create a Revolved Surface

The sample code in the file UgSktRevolveSurface.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a revolved surface. It includes all possible element assignments. By following the instructions for the feature you want to create, it is possible to remove element settings not appropriate for your use.

The Element Tree for First Features

First features (extrude and revolve solids created as the first feature in a part) require a subset of the standard element tree and some special handling for the section pointer.

The following table lists the elements applicable to first feature creation (extrude or revolve):

Element ID	Value
PRO_E_FEATURE_TYPE	Feature type: PRO_FEAT_FIRST
PRO_E_FEATURE_FORM	PRO_EXTRUDE / PRO_REVOLVE
PRO_E_FEAT_FORM_IS_THIN	Feature Form Of Type ProRevFeatForm = PRO_REV_FEAT_FORM_NO_THIN for a Solid feature = PRO_REV_FEAT_FORM_THIN for a Thin feature
PRO_E_BODY_USE	Mandatory. Specifies the body to add geometry to. The valid values are: • PRO_BODY_USE_NEW—The geometry in the feature is stored in the new body. • PRO_BODY_USE_SELECTED—The geometry in the feature is stored in the single selected body.
PRO_E_BODY_SELECT	Specifies the reference to the selected body. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED **Note** Only single reference is allowed.
PRO_E_SKETCHER	Sketcher pointer. Used because the standard section requires selected references not available in an empty model.
PRO_E_STD_MATRLSIDE*	Mandatory if thin Of type ProExtMatlSide (Extrude) Of type ProRevMatlSide (Revolve)
PRO_E_THICKNESS	Mandatory >= 0.0 if thin Of type PRO_VALUE_TYPE_DOUBLE
PRO_E_EXT_DEPTH_FROM	Compound Element (Extrude only)
PRO_E_EXT_DEPTH_FROM_VAL	Depth dimension (of type PRO_VALUE_TYPE_DOUBLE) (Extrude only)
PRO_E_REV_ANGLE_FROM	Compound Element (Revolve only)

Element ID	Value
PRO_E_REV_ANGLE_FROM_VAL	Angular dimension (of type PRO_VALUE_TYPE_DOUBLE) (Revolve only)
PRO_E_STD_FEATURE_NAME	Default given by application depending on the feature type. Can be modified by the user.

Elements identified with '*' depend on the definition of the standard section. These elements may not be assigned values until the standard section has been completely allocated (which typically happens during redefine of the feature). Values assigned to these elements while the section is not complete are ignored.

Example 9: Creating the First Extruded Protrusion Feature by Conventional Approach

The sample code in the file `UgSktFirstFeatureCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create the first extruded protrusion using the approach for the sketched features.

The following example:

- Creates an incomplete feature using `ProFeatureCreate()`,
- Extracts the section from the element tree of the incomplete feature,
- Builds the section on the section handle obtained, and,
- Completes the feature using `ProFeatureRedefine()`.

Following is the change in the approach for Pro/ENGINEER Wildfire release:

1. Level of `PRO_E_SKETCHER` in an element tree is changed.

   For any Pro/ENGINEER release previous to Wildfire:
   ```
   PRO_E_FEATURE_TREE -> PRO_E_STD_SECTION -> PRO_E_SKETCHER
   ```

   For Pro/ENGINEER Wildfire release:
   ```
   PRO_E_FEATURE_TREE -> PRO_E_SKETCHER
   ```

2. Value of `PRO_E_SKETCHER`—A new `ProValue` is to be allocated and then assigned to the element (rather than the old approach of reusing the value extracted from the element tree).

Example 10: Creating the First Thin Revolve Protrusion Feature by Conventional Approach

The sample code in the file `UgSktFirstFeatureRevCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a thin first revolve Protrusion using the approach for sketched features.

The following example:

- Creates an incomplete feature using `ProFeatureCreate()`,
- Extracts the section from the element tree of the incomplete feature,
- Builds the section on the section handle obtained, and,
- Completes the feature using `ProFeatureRedefine()`.

Following is the change in the approach for Pro/ENGINEER Wildfire release:

1. Level of `PRO_E_SKETCHER` in an element tree is changed.

   For any Pro/ENGINEER release previous to Wildfire:
   ```
   PRO_E_FEATURE_TREE -> PRO_E_STD_SECTION -> PRO_E_SKETCHER
   ```

   For Pro/ENGINEER Wildfire release:
   ```
   PRO_E_FEATURE_TREE -> PRO_E_SKETCHER
   ```

2. Value of `PRO_E_SKETCHER`—A new ProValue is to be allocated and then assigned to the element (rather than the old approach of reusing the value extracted from the element tree).

46

Element Trees: Ribs

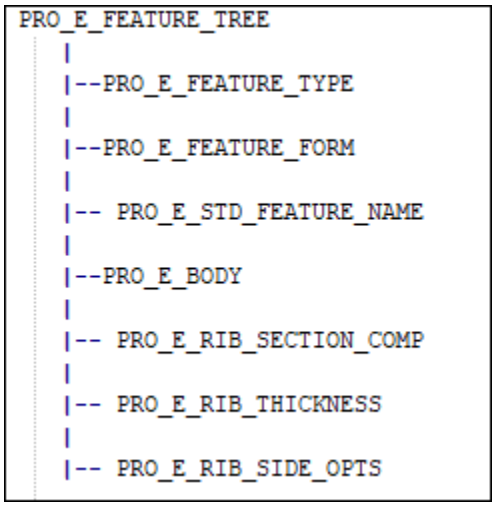
This chapter describes how to use the include files `ProRib.h`, so that you can create ribs programmatically. As ribs are sketched features; we recommend you to read the chapters Element Trees: Principles of Feature Creation on page 764 and Element Trees: Sketched Features on page 1004 before referring to this chapter.

The Element Tree for Rib Features

The element tree for Rib features is documented in the Creo Parametric header file `ProRib.h`.

The following figure shows the element tree for rib features.

The Element Tree for Rib Feature



The following table lists the data types for rib feature and their permissible values:

Element ID	Data Type	Description
`PRO_E_FEATURE_TYPE`	`PRO_VALUE_TYPE_INT`	Feature type: `PRO_FEAT_RIB`
`PRO_E_FEATURE_FORM`	`PRO_VALUE_TYPE_INT`	Mandatory= `PRO_EXTRUDE`
`PRO_E_STD_FEATURE_NAME`	`PRO_VALUE_TYPE_WSTRING`	Name of the rib feature. The default value is `RIB`.
`PRO_E_BODY`	Compound	Compound element that holds Body options. For more information, refer to the `ProBodyOpts.h` element tree.
`PRO_E_BODY_USE`	`PRO_VALUE_TYPE_INT`	Holds Body use options. The valid value is `PRO_BODY_USE_SELECTED`.
`PRO_E_BODY_SELECT`	`PRO_VALUE_TYPE_SELECTION`	Specifies the reference to the selected body.

Element ID	Data Type	Description
		Note Only single reference is allowed.
PRO_E_RIB_SECTION_COMP	Compound Element	Holds section related elements.
PRO_E_RIB_THICKNESS	PRO_VALUE_TYPE_DOUBLE	Thickness of the rib. The value of the thickness should be positive and bigger than zero. The default value depends on part epsilon.
PRO_E_STD_SECTION	PRO_VALUE_TYPE_INT	Compound Element

Element ID	Data Type	Description
PRO_E_STD_MATRLSIDE	PRO_VALUE_TYPE_INT	Material side options of the rib: • PRO_RIB_SEC_SIDE_ONE—Rib will be on side one of the section • PRO_RIB_SEC_SIDE_TWO—Rib will be on side two of the section **Note** The element PRO_E_STD_MATRLSIDE is directly dependent upon the presence of a fully defined PRO_E_STD_SECTION element tree. Any value assigned to this element before fully defining the PRO_E_STD_SECTION, will be ignored. The default value depends on the sketch under the PRO_E_STD_SECTION element.
PRO_E_RIB_SIDE_OPTS	PRO_VALUE_TYPE_INT	Side options of the rib: • PRO_RIB_SYMMETRIC— Rib will be symmetric to the sketch. This is the default value. PRO_RIB_SIDE_ONE—Rib will be on Side One of the sketch.

Element ID	Data Type	Description
		• PRO_RIB_SIDE_TWO—Rib will be on Side Two of the sketch.

47

Element Trees: Sweep

This chapter describes the principles of creating a sweep feature. The chapters Element Trees: Principles of Feature Creation on page 764 and Element Trees: Sketched Features on page 1004 provide the necessary background for this topic. Read those chapters before this one.

Sweeps in Creo Parametric TOOLKIT

Creo Parametric provides access and creates a constant section sweep feature. You can create a solid or surface feature. You can also add or remove material while sweeping a section along one or more selected trajectories by controlling the section's orientation, rotation, and geometry.

Sweep Feature

The element tree for the sweep feature is documented in the header file `ProSweep.h`, and is shown in the following figure.

Element Tree for Sweep Feature

```
PRO_E_FEATURE_TREE
 |
 |---PRO_E_FEATURE_FORM
 |---PRO_E_SWEEP_TYPE
 |
 |---PRO_E_SWEEP_FRAME_COMP
 |
 |---PRO_E_SWEEP_PROF_COMP
 |     |---PRO_E_SWP_SEC_TYPE
 |     |---PRO_E_SWEEP_SECTION
 |           |---PRO_E_SKETCHER  POINTER
 |---PRO_E_SWP_ATTR
 |     |---PRO_E_END_SRF_ATTR
 |
 |---PRO_E_STD_FEATURE_NAME
 |---PRO_E_EXT_SURF_CUT_SOLID_TYPE
 |---PRO_E_REMOVE_MATERIAL
 |---PRO_E_FEAT_FORM_IS_THIN
 |---PRO_E_STD_MATRLSIDE
 |---PRO_E_THICKNESS
 |---PRO_E_TRIM_QUILT
 |---PRO_E_TRIM_QLT_SIDE
 |---PRO_E_BODY
 |     |--PRO_E_BODY_USE
 |     |--PRO_E_BODY_SELECT
```

The following table describes the elements in the element tree for sweeps:

Element ID	Data Type	Description
`PRO_E_FEATURE_FORM`	`PRO_VALUE_TYPE_INT`	Specifies the feature form. The valid value is `PRO_SWEEP` defined in the enumerated data type `ProFeatFormType` in `ProFeatForm.h`.
`PRO_E_SWEEP_TYPE`	`PRO_VALUE_TYPE_INT`	Specifies the type of sweep. The valid value is `PRO_SWEEP_`

Element ID	Data Type	Description
		TYPE_MULTI_TRAJ defined in the enumerated data type ProSweepType.
PRO_E_SWEEP_FRAME_COMP	Compound	This compound element specifies the trajectories and orientation of the section plane. For more information on this element, refer to the section Element Tree for PRO_E_SWEEP_FRAME_COMP on page 1046.
PRO_E_SWEEP_PROF_COMP	Compound	This compound element defines the sweep type and holds the sketch related elements for the sweep cross section.
PRO_E_SWP_SEC_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of sweep. The valid value is PRO_SWEEP_CONST_SECTION defined in the enumerated data type ProSweepSecType. The option specifies a constant section sweep.
PRO_E_SWEEP_SECTION	Compound	This compound element contains the sketch for the sweep cross section.
PRO_E_SKETCHER	PRO_VALUE_TYPE_POINTER	Specifies a sketcher pointer. The user defined sketch can be directly retrieved from this element.
PRO_E_SWP_ATTR	Compound	This compound element defines the sweep attributes. The element is available only for surface sweeps.
PRO_E_END_SRF_ATTR	PRO_VALUE_TYPE_INT	Specifies the option to keep the end of sweep feature open or capped. The enumerated data type ProSweepEndSrfAttr contains the valid values for this element: • PRO_SWEEP_END_SRF_OPEN • PRO_SWEEP_END_SRF_CAPPED
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is SWEEP.
PRO_E_EXT_SURF_CUT_SOLID_TYPE	PRO_VALUE_TYPE_INT	Specifies a solid or surface feature type. The enumerated data type ProSweepFeatType contains the following valid values for this element: • PRO_SWEEP_FEAT_TYPE_SOLID • PRO_SWEEP_FEAT_TYPE

Element ID	Data Type	Description
		_SURFACE
PRO_E_REMOVE_MATERIAL	PRO_VALUE_TYPE_INT	Specifies whether to extrude or remove the material along the sweep. The enumerated data type ProSweepRemMaterial contains the following valid values for this element: • PRO_SWEEP_MATERIAL_ADD • PRO_SWEEP_MATERIAL_REMOVE
PRO_E_FEAT_FORM_IS_THIN	PRO_VALUE_TYPE_INT	Specifies whether to create a thin section. The enumerated data type ProSweepFeatForm contains the following valid values for this element: • PRO_SWEEP_FEAT_FORM_NO_THIN • PRO_SWEEP_FEAT_FORM_THIN
PRO_E_STD_MATRLSIDE	PRO_VALUE_TYPE_INT	Specifies the direction in which material will be added with respect to the sketch. This element is mandatory for all cuts and thin features. The enumerated data type ProSweepMatlSide contains the following valid values for this element: • PRO_SWEEP_MATERIAL_SIDE_ONE • PRO_SWEEP_MATERIAL_SIDE_TWO • PRO_SWEEP_MATERIAL_BOTH_SIDES
PRO_E_THICKNESS	PRO_VALUE_TYPE_DOUBLE	Specifies the thickness for the thin feature. This element is mandatory for thin features.
PRO_E_TRIM_QUILT	PRO_VALUE_TYPE_SELECTION	Specifies the selection of quilt to be trimmed.
PRO_E_TRIM_QLT_SIDE	PRO_VALUE_TYPE_INT	Specifies the side of the quilt to be trimmed. This element is relevant when the element PRO_E_STD_MATRLSIDE has its value as PRO_SWEEP_MATERIAL_BOTH_SIDES. The enumerated data type ProSweepTrimQltSide contains the following valid values for this element:

Element ID	Data Type	Description
		• PRO_SWEEP_TRIMQLT_SIDE_ONE • PRO_SWEEP_TRIMQLT_SIDE_TWO
PRO_E_BODY	Compound element	Compound element for body options.
PRO_E_BODY_USE	PRO_VALUE_TYPE_INT	Mandatory. Specifies the body to add geometry to. Defined by the enumerated data type ProBodyUseOpts and the valid values follow: • PRO_BODY_USE_NEW—Feature stores its geometry in the new body. • PRO_BODY_USE_SELECTED—Feature adds its geometry to single selected body.
PRO_E_BODY_SELECT	PRO_VALUE_TYPE_SELECTION	Specifies the reference to the selected bodies. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED.

For elements specific to sheetmetal, refer to the chapter Production Applications: Sheetmetal on page 1310.

Element Tree for PRO_E_SWEEP_FRAME_COMP

The element PRO_E_SWEEP_FRAME_COMP is a compound element, which defines the sweep type and holds the sketch related elements for the sweep cross section. The element tree for PRO_E_SWEEP_FRAME_COMP is as shown in the figure below:

```
|---PRO_E_SWEEP_FRAME_COMP
|     |---PRO_E_FRM_OPT_TRAJ
|     |     |---PRO_E_OPT_TRAJ
|     |          |---PRO_E_STD_SEC_METHOD
|     |          |
|     |          |---PRO_E_STD_SEC_SELECT
|     |               |---PRO_E_STD_CURVE_COLLECTION_APPL
|     |
|     |
|     |---PRO_E_FRAME_SETUP
|     |     |---PRO_E_FRM_NORMAL
|     |     |---PRO_E_FRM_PIVOT_DIR
|     |     |     |---PRO_E_DIRECTION_COMPOUND
|     |     |          |---PRO_E_DIRECTION_REFERENCE
|     |     |          |---PRO_E_DIRECTION_FLIP
|     |     |---PRO_E_FRM_CONST_Z
|     |     |     |---PRO_E_DIRECTION_COMPOUND
|     |     |          |---PRO_E_DIRECTION_REFERENCE
|     |     |          |---PRO_E_DIRECTION_FLIP
|     |     |---PRO_E_FRM_ORIENT
|     |     |---PRO_E_FRM_NORM_SURF    (COMPOUND)
|     |     |     |---PRO_E_SURF_CHAIN_CMPND
|     |     |     |     |---PRO_E_SURF_CHAIN_METHOD
|     |     |     |     |---PRO_E_SURF_CHAIN_REF_SURFS
|     |     |     |---PRO_E_FRM_NORM_SURF_SIDE
|     |     |---PRO_E_FRM_USER_X
|     |     |---PRO_E_FRM_START_X
|     |     |     |---PRO_E_DIRECTION_COMPOUND
|     |     |          |---PRO_E_DIRECTION_REFERENCE
|     |     |          |---PRO_E_DIRECTION_FLIP
```

The following table describes the sub elements of the PRO_E_SWEEP_FRAME_COMP:

Element ID	Data Type	Description
PRO_E_FRM_OPT_TRAJ	Array	Specifies an array of single trajectory that is only one element.
PRO_E_OPT_TRAJ	Compound	This compound element specifies the trajectory.
PRO_E_STD_SEC_METHOD	PRO_VALUE_TYPE_INT	Specifies that the trajectory is selected. The valid value is PRO_SEC_SELECT defined in the enumerated data type ProSecMethod in ProStdSection.h.
PRO_E_STD_SEC_SELECT	Compound	This compound element specifies the collection of trajectory.
PRO_E_STD_CURVE_	PRO_VALUE_TYPE_	Specifies the collection of curves

Element ID	Data Type	Description
COLLECTION_APPL	SELECTION	and edge chains for the trajectory.
PRO_E_FRAME_SETUP	Compound	This compound element specifies the orientation of the section plane.
PRO_E_FRM_NORMAL	PRO_VALUE_TYPE_INT	Specifies the orientation of the section plane normal. The enumerated data type ProFrameNormal contains the valid values for this element: • PRO_FRAME_NORM_ORIGIN—The section plane normal is perpendicular to the origin trajectory throughout the sweep. • PRO_FRAME_PIVOT_DIR—The z-axis is tangent to the projection of the origin trajectory at the direction specified in the element PRO_E_FRM_PIVOT_DIR. • PRO_FRAME_CONST_Z_DIR —The section plane remains parallel to the specified direction reference vector.
PRO_E_FRM_PIVOT_DIR	Compound	This compound element for the direction reference is relevant when the element PRO_E_FRM_NORMAL has its value as PRO_FRAME_PIVOT_DIR.
PRO_E_DIRECTION_COMPOUND	Compound	This compound element specifies the direction reference for PRO_E_FRM_PIVOT_DIR. The compound element is a standard Creo Parametric element subtree and is described in ProDirection.h.
PRO_E_FRM_CONST_Z	Compound	This compound element for the direction reference is relevant when the element PRO_E_FRM_NORMAL has its value as PRO_FRAME_CONST_Z_DIR.
PRO_E_DIRECTION_COMPOUND	Compound	This compound element specifies the direction reference for PRO_FRAME_CONST_Z_DIR. The compound element is a standard Creo Parametric element subtree and is described in ProDirection.h.
PRO_E_FRM_ORIENT	PRO_VALUE_TYPE_INT	Specifies how the rotation of the frame around the sketch plane's normal is oriented along the

Element ID	Data Type	Description
		sweep. The enumerated data type ProFrameOrient contains the valid vales for this element: • PRO_FRAME_MIN—Corresponds to the **Automatic** option in Creo Parametric user interface. The direction of the x-vector is calculated so that the swept geometry is minimally twisted. • PRO_FRAME_NORM_SURF—Orients the y-axis of the section plane to be normal to the surface on which the origin trajectory lies. • PRO_FRAME_CONSTANT—This option must be set when the element PRO_E_FRM_NORMAL has its value as PRO_FRAME_PIVOT_DIR. • PRO_FRAME_X_TRAJ—This option is not supported as Creo Parametric supports only single trajectory in sweep feature.
PRO_E_FRM_NORM_SURF	Compound	This compound element for the edge chains is relevant when the element PRO_E_FRM_ORIENT has its value as PRO_E_FRAME_NORM_SURF.
PRO_E_SURF_CHAIN_CMPND	Compound	This compound element specifies the surface edge collection and the collection method.
PRO_E_SURF_CHAIN_METHOD	PRO_VALUE_TYPE_INT	Specifies the collection method for the surface edges. The valid values for this element are: • PRO_SURF_CHAIN_METHOD_DEFAULT1 • PRO_SURF_CHAIN_METHOD_DEFAULT2 **Note** Both the options are available only if the trajectory is a two-sided edge of a surface. In case of one-sided edge, only one option is available.
PRO_E_SURF_CHAIN_REF_SURFS	Array	Specifies an array of surface edges.

Element ID	Data Type	Description
PRO_E_SURF_CHAIN_SURF	Compound	This compound element specifies the surface edge collection.
PRO_E_SURF_CHAIN_REF	PRO_VALUE_TYPE_SELECTION	Specifies the collection of two-sided edge of a surface.
PRO_E_FRM_NORM_SURF_SIDE	PRO_VALUE_TYPE_INT	This element is relevant when the element PRO_E_FRM_ORIENT has its value as PRO_E_FRAME_NORM_SURF. Specifies the normal surface direction. The enumerated data type ProFrmNormSrfSide contains the valid vales for this element: • PRO_FRAME_NORM_SRF_SIDE_INSIDE • PRO_FRAME_NORM_SRF_SIDE_OUTSIDE
PRO_E_FRM_USER_X	PRO_VALUE_TYPE_INT	This element is relevant when the element PRO_E_FRM_ORIENT has its value as PRO_FRAME_MIN. Specifies if the direction of x-axis at start is automatically computed or is user defined. The enumerated data type ProFrameStartX contains the valid values for this element: • PRO_FRAME_DEFAULT_START_X • PRO_FRAME_USER_START_X **Note** It is sometimes necessary to specify the x-axis direction, for example, for straight line trajectories or trajectories that have a straight segment at the start.

Element ID	Data Type	Description
PRO_E_FRM_START_X	Compound	This compound element for the direction reference is relevant when the element PRO_E_FRM_USER_X has its value as PRO_FRAME_USER_START_X.
PRO_E_DIRECTION_COMPOUND	Compound	This compound element specifies the direction reference for PRO_FRAME_USER_START_X. The compound element is a standard Creo Parametric element subtree and is described in ProDirection.h.

Creating a Sweep Feature

To Create a Sweep Feature

1. Create an incomplete feature without the PRO_E_SKETCHER element.
2. Call ProFeatureElemtreeExtract() with the feature handle to get the new feature tree. This results in an initialized PRO_E_SKETCHER element and sketch handle.
3. Create the section with the initialized sketch handle.
4. Call ProFeatureWithoptionsRedefine() to put the section in the Creo Parametric database to complete the sweep feature.

Note

If these elements PRO_E_FRAME_SETUP and PRO_E_FRM_ORIENT are not defined, you have to redefine the feature twice to get the valid initiated sketch handle.

Example 1: Creating a Sweep Feature

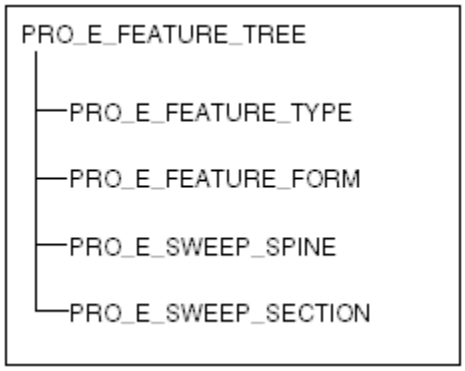
The sample code in UgCreoSweepCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a sweep feature.

Simple Sweep Feature

The simple sweep element tree documented in the header file `ProSweep.h` is obsolete. Use the sweep element tree instead to create and access the SWEEP feature.

The element tree for the simple sweep feature is shown in the following figure.

Element Tree for Simple Sweep Feature



The following table describes the elements in the element tree for simple sweeps:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Feature type
PRO_E_FEATURE_FORM	PRO_VALUE_TYPE_INT	Feature form (PRO_SWEEP)
PRO_E_SWEEP_SPINE	Compound	Trajectory (like (PRO_E_STD_SECTION)
PRO_E_SWEEP_SECTION	Compound	Section (like PRO_E_STD_SECTION)

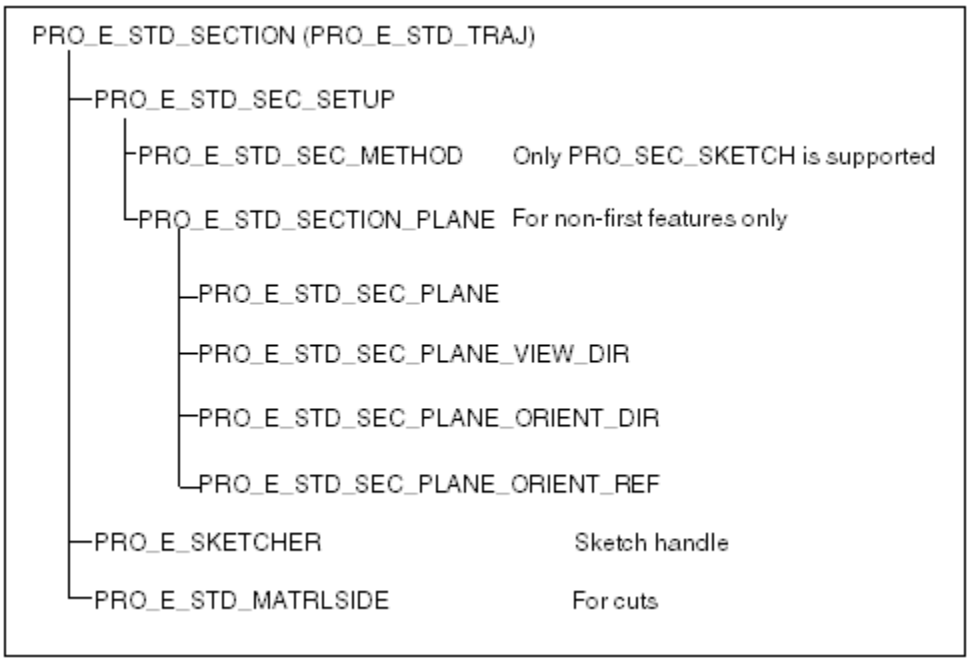
The element tree definitions of `PRO_E_SWEEP_SPINE` and `PRO_E_SWEEP_SECTION` take on the same form as the element `PRO_E_STD_SECTION` (documented in the header file `ProStdSection.h`).

Note

Release 20 of Pro/TOOLKIT supports only sketched, constant cross-sectional sweeps.

The following figure shows the valid elements within this subtree.

Element Subtree for Simple Sweep



Swept, constant, cross-sectional feature forms are supported for the following feature types:

- `PRO_FEAT_FIRST_FEAT`
- `PRO_FEAT_PROTRUSION`
- `PRO_FEAT_CUT`
- `PRO_FEAT_DATUM_SURF`

Creating a Simple Sweep Feature

To Create a Simple Sweep Feature

1. Create an incomplete feature with the `PRO_E_FEATURE_TYPE` and `PRO_E_FEATURE_FORM` elements defined. Also define the compound elements `PRO_E_SWEEP_SPINE` and `PRO_E_SWEEP_SECTION`, down to the `PRO_E_STD_SEC_METHOD` element (see the subtree for details).
2. Call `ProFeatureElemtreeExtract()` with the feature handle to get the new feature tree. This results in an initialized `PRO_E_SWEEP_SPINE` subtree and sketch handle.
3. Create the spine section with the initialized sketch handle.
4. Call `ProFeatureWithoptionsRedefine()` as an incomplete feature to put the spine section in the Creo Parametric database.

5. Call `ProFeatureElemtreeExtract()` to get the new feature tree. This results in an initialized `PRO_E_SWEEP_SECTION` subtree and sketch handle. This step is necessary because the sweep section is dependent on the spine section.
6. Create the sweep section with the initialized sketch handle. This section automatically contains the centerline cross hairs of the sweep section.

   The cross hairs can be used to locate and dimension the section.
7. Call `ProFeatureWithoptionsRedefine()` with any option except incomplete to complete the simple sweep feature.

Example 1: Creating a Simple Sweep First Feature Protrusion

The sample code in `UgSweepCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a simple sweep first feature protrusion using the Creo Parametric functions.

Example 2: Creating a Simple Sweep Protrusion Feature by Conventional Approach

The sample code in `Ug3DSweepCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` shows how to create a simple sweep protrusion feature by the conventional approach for the sketched features.

The user is prompted to select the following:

- Sketching plane
- Orientation plane
- Orthogonal edge for the dimensioning of the spine (trajectory - `PRO_E_SWEEP_SPINE`) section
- Orthogonal edge for the sweep section (`PRO_E_SWEEP_SECTION`)

48

Element Trees: Solid Body

This chapter describes the element tree structure for body options that can be selected while creating a feature. You can perform geometric operations such as splitting a body or merging with other bodies and boolean operations such as merge, intersect, and subtract.

The following sections describe the procedure for performing Boolean operations such as intersect, subtract, and, merge bodies in a model. This chapter also describes how to copy, remove, and split bodies.

Introduction

Using Creo Parametric TOOLKIT, you can create bodies. The creation and manipulation of bodies use the following Creo Parametric TOOLKIT objects:

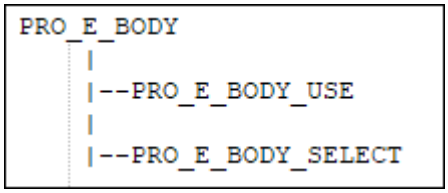
- ProSolid
- ProSolidBody

The Element Tree for Body Options

The element tree for Body options is documented in the Creo Parametric header file `ProBodyOpts.h`.

The following figure shows the element tree for body options feature.

The Element Tree for Body Options Feature



The following table describes the elements in the element tree:

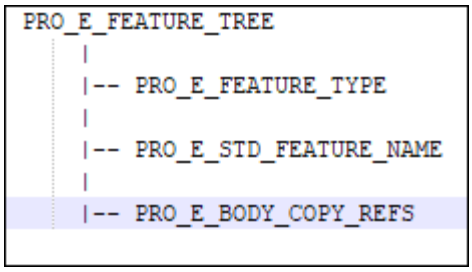
Element ID	Data Type	Description
PRO_E_BODY	Compound element	Compound element for body options.
PRO_E_BODY_USE	PRO_VALUE_TYPE_INT	Mandatory. Specifies the body to add geometry to. Defined by the enumerated data type ProBodyUseOpts and the valid values follow: • PRO_BODY_USE_NEW— Feature stores its geometry in the new body. • PRO_BODY_USE_ALL—Feature cuts from all existing bodies. • PRO_BODY_USE_SELECTED—Feature adds or removes its geometry to or from selected bodies.
PRO_E_BODY_SELECT	PRO_VALUE_TYPE_SELECTION	Specifies the reference to the selected bodies. Mandatory if PRO_E_BODY_USE is set to PRO_BODY_USE_SELECTED.

The Element Tree for Body Copy Feature

You can perform copy operation on bodies. When you copy bodies in a part, a new body is created for each source body. The element tree for the body copy feature is documented in the Creo Parametric header file ProBodyCopy.h.

The Element Tree for Body Copy Feature

The following figure shows the element tree for the body copy feature.



The following table describes the elements in the element tree:

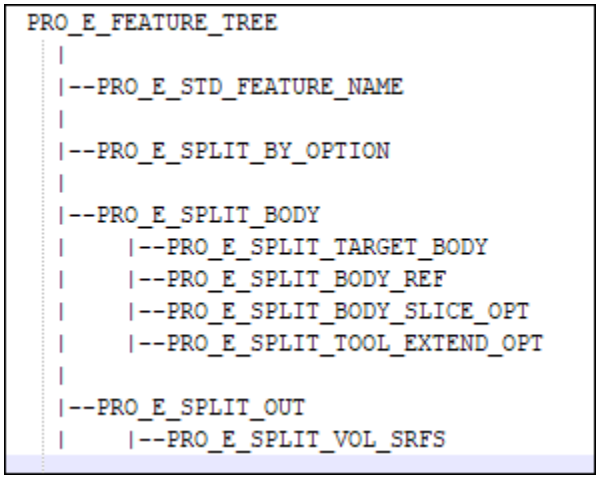
Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the feature type. Valid value is PRO_FEAT_BODY_COPY.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the feature name.
PRO_E_BODY_COPY_REFS	PRO_VALUE_TYPE_SELECTION	Mandatory. Specifies the bodies to copy. Only bodies from the same model are allowed. **Note** Use the function ProElementReferencesSet(), if you need to set multiple reference values for PRO_E_BODY_COPY_REFS.

The Element Tree for Body Split Feature

You can split a body into two bodies. The geometry of the original body is divided between the original body and the new body. The element tree for the body split feature is documented in the Creo Parametric header file ProSplitbody.h.

The Element Tree for Body Split Feature

The following figure shows the element tree for the split body feature.



The following table describes the elements in the element tree:

Element ID	Data Type	Description
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Name of the feature.
PRO_E_SPLIT_BY_OPTION	PRO_VALUE_TYPE_INT	Mandatory element. Split by option.
PRO_E_SPLIT_BODY	Compound element	Split by object.
PRO_E_SPLIT_TARGET_BODY	PRO_VALUE_TYPE_SELECTION	Target body to be split.
PRO_E_SPLIT_BODY_REF	PRO_VALUE_TYPE_SELECTION	Split by body reference.
PRO_E_SPLIT_BODY_SLICE_OPT	PRO_VALUE_TYPE_INT	Split by slice direction.
PRO_E_SPLIT_TOOL_EXTEND_OPT	PRO_VALUE_TYPE_INT	Split by extended object.
PRO_E_SPLIT_OUT	Compound element	Split by volume.
PRO_E_SPLIT_VOL_SRFS	PRO_VALUE_TYPE_SELECTION	Split by surfaces.

Splitting by Object

You can split one body into two bodies using a plane, surface, or quilt as the splitting object. The following table lists the mandatory element types for splitting a body by object:

Element ID	Data Type
PRO_E_SPLIT_BY_OPTION	The valid value is PRO_SPLIT_BY_SPLITTING_OBJ of the enumerated data type ProSplitByType.
PRO_E_SPLIT_TARGET_BODY	Target bodies to be split.
PRO_E_SPLIT_BODY_REF	Split by body reference.
PRO_E_SPLIT_BODY_SLICE_OPT	Side 1 / Side 2 of the enumerated data type ProSplitBodySliceOpt.
PRO_E_SPLIT_TOOL_EXTEND_OPT	Extend option of the enumerated data type ProSplitBodyToolExtendOpt.

Splitting by Volume type

When the geometry of the original body has disjoint volumes, you can split out one or more of these volumes into a new body. Select one or more body surfaces to define that volume as a new body. The following table lists the mandatory element types for splitting a body by volume type:

Element ID	Data Type
PRO_E_SPLIT_BY_OPTION	The valid value is PRO_SPLIT_BY_VOLUME of the enumerated data type ProSplitByType.
PRO_E_SPLIT_VOL_SRFS	Split by surfaces or surface regions.

A body can be divided into two bodies by an intersecting object or by volume. The enumerated data type ProSplitByType defines the way the body is split and has the following valid values:

- PRO_SPLIT_BY_SPLITTING_OBJ—A body can be split by an intersecting object.
- PRO_SPLIT_BY_VOLUME—A body that has more than one disjoint volume can be split by picking up the surface or surface regions of the desired portion to split out as a new body.

The splitting side of the body is defined by the enumerated data type ProSplitBodySliceOpt. When a single body is split, an additional body is created. The valid values follow:

- PRO_E_SPLIT_BODY_SLICE_FIRST_OPT—Side 1 is the default option and the value is 0.
- PRO_E_SPLIT_BODY_SLICE_SECOND_OPT—Flips the default body creation and creates a body in the other side.

The extend object is available for splitting by object and is defined by the enumerated data type `ProSplitBodyToolExtendOpt` and has the following valid values:

- `PRO_E_SPLIT_TOOL_EXTEND_OPT_NO`—The splitting object intersects the body graphically.
- `PRO_E_SPLIT_TOOL_EXTEND_OPT_YES`—The splitting object is extended to intersect with the bodies. The extend option is available for the geometrically extendable objects such as a plane.

The Element Tree for Body Remove Feature

As part of multibody design, there could be bodies that you want to remove from the model. For example, you created bodies as tools in the design process and you do not need them in the final model, or the system created bodies when you imported geometry from other models. You cannot delete these bodies, because they have contributing features. In such cases, you can remove the bodies.

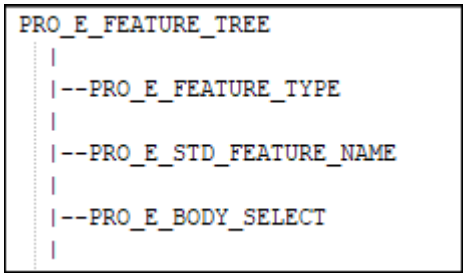
Note

Bodies that are removed do not contribute to mass properties.

The element tree for the body remove feature is documented in the Creo Parametric header file `ProRemoveBody.h`. Using this element tree you can remove bodies in a model.

The Element Tree for Body Remove Feature

The following figure shows the element tree for the body remove feature.



The following table describes the elements in the element tree:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the feature type. The valid value is PRO_FEAT_REMOVEBODY.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the feature name. Default value is Remove body.
PRO_E_BODY_SELECT	PRO_VALUE_TYPE_SELECTION	Mandatory. Specifies the references for the bodies that needs to be removed. The valid selection reference is from PRO_BODY type only. **Note** Use the function ProElementReferencesSet(), if you need to set multiple reference values for PRO_E_BODY_SELECT.

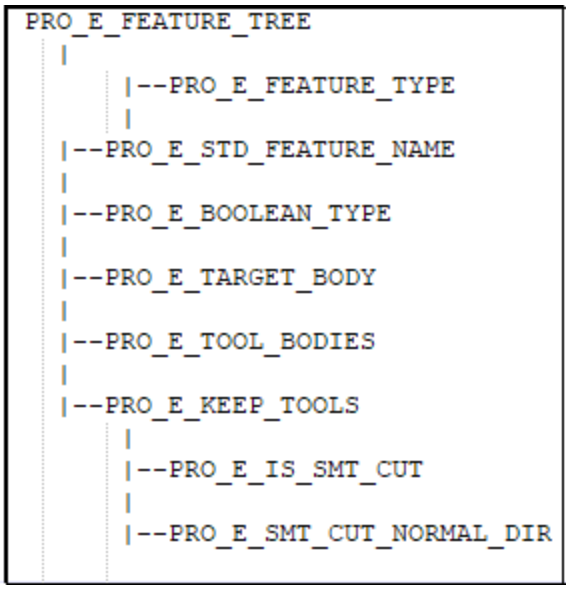
The Element Tree for Boolean Body Operations

When a part contains more than one body, you can use the Boolean Operations feature to perform geometric operations such as **Merge**, **Intersect**, and **Subtract**.

The element tree for Boolean body operations is documented in the Creo Parametric header file ProBooleanBodies.h. You can perform subtract, merge, and intersect operations on bodies.

The Element Tree for The Element Tree for Boolean Body Operation

The following figure shows the element tree for Boolean body operation.



The following table describes the elements in the element tree:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the feature type. The valid value is PRO_FEAT_BOOLEANBODIES.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the feature name for combined Boolean body. Default value depends on the operation type: • Body Merge • Body Intersect • Body Subtract
PRO_E_BOOLEAN_TYPE	PRO_VALUE_TYPE_INT	Specifies the Boolean operation type that needs to be performed on the bodies and is defined by the enumerated data type ProBooleanbodies TypeOption. The valid values are: • MERGE_BOOL_TYPE—Body Merge • INTERSECT_BOOL_

Element ID	Data Type	Description
		TYPE—Body Intersect • SUBTRACT_BOOL_TYPE—Body Subtract
PRO_E_TARGET_BODY	PRO_VALUE_TYPE_SELECTION	Specifies the reference to the modified bodies. Mandatory element and can be filled with single body MERGE_BOOL_TYPE or INTERSECT_BOOL_TYPE or multiple bodies SUBTRACT_BOOL_TYPE . The valid selection reference is from PRO_BODY type only. **Note** Use the function ProElementReferencesSet(), if you need to set multiple reference values for PRO_E_TARGET_BODY.
PRO_E_TOOL_BODIES	PRO_VALUE_TYPE_SELECTION	Specifies the reference to the modifying bodies. Mandatory element and can be filled with single body SUBTRACT_BOOL_TYPE or multiple bodies MERGE_BOOL_TYPE or INTERSECT_BOOL_TYPE. The valid selection reference is from PRO_BODY type only.
PRO_E_KEEP_TOOLS	PRO_VALUE_TYPE_INT	This element type is available only for SUBTRACT_BOOL_

Element ID	Data Type	Description
		TYPE Boolean body operation. It specifies the body options that need to be kept. This element type is defined by the enumerated data type ProBooleanbodiesKeepBodyOption and the valid values are: • KEEP_TOOL_NO—default value. Deletes the modifying bodies. • KEEP_TOOL_YES—keeps the modifying bodies.
PRO_E_IS_SMT_CUT	PRO_VALUE_TYPE_INT	This element is applicable in sheetmetal parts only and it controls the cut type as follows: • PRO_B_TRUE—SMT cut type • PRO_B_FALSE—Solid cut type

Element ID	Data Type	Description
PRO_E_SMT_CUT_NORMAL_DIR	PRO_VALUE_TYPE_INT	This element is applicable in sheetmetal parts only and it controls the SMT cut geometry driving surface. The element type is defined by the enumerated data type ProSmtCutNormDir and the valid values are: • PRO_SMT_CUT_DRVSIDE_GREEN—For normal to Driven surface. • PRO_SMT_CUT_DRVSIDE_WHITE—For normal to Offset surface.

49

Element Trees: Creo Flexible Modeling Features

This chapter describes how to construct and access the element tree for some Creo Flexible Modeling features in Creo Parametric TOOLKIT. It also shows how to redefine, create and access the properties of these features.

Move and Move-Copy Features

This section describes how to construct and access the element tree for Move and Move-Copy features. It also shows how to create, redefine, and access the properties of these features.

Introduction

The Move and the Move-Copy features allow a rigid transformation to a geometry selection or its copy. The surfaces within the geometry selection must belong either to the solid geometry or to a single quilt. You can move the following the geometry selection:

- Any surface collection within the solid geometry or a single quilt.
- An intent surface within the solid geometry or a single quilt.
- Regular or intent datums (planes, axes, points and coordinate systems).
- Regular or intent curves.
- Any combination of the above geometries.

When you create a move feature at assembly level, the following types of references can be moved:

- Geometry of assembly components—Part level geometry only, that is, surfaces, quilts, curves, datums
- Assembly components—Parts and subassemblies

The moved entities are copies of the original entities and will have new IDs. The original entities can be removed (Move) or kept (Copy-Move).

A Move feature will act on a single set of objects. To move different geometry selections, multiple move features must be created.

The Element Tree for Move and Move-Copy

The element tree for the Move and Move-Copy feature is documented in the header file `ProFlexMove.h`, and is shown in the following figure:

Element Tree for Move and Move-Copy

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_FLEXMOVE_MOVED_GEOMETRY
    |
    |-- PRO_E_FLEXMOVE_MOVED_COMPS
    |
    |-- PRO_E_FLEXMOVE_DEFINE_METHOD
    |
    |-- PRO_E_D3ELEM_SETS (D3Elem branch, see ProD3Elem.h)
    |
    |-- PRO_E_FLEXMOVE_DIMS_COMPOUND
    |
    |-- PRO_E_FLEX_OPTS_CMPND     (General FLX-MDL options branch,
    |                              see ProFlxmdlOpts.h)
    |-- PRO_E_FLXSLV_PROP_CONSTRS (General FLX-MDL tangency
    |                              propagation branch,
    |                              see ProFlexTanPropOpts.h)
    |-- PRO_E_COMPONENT_MISC_ATTR
    |
    |-- PRO_E_COMPONENT_INIT_POS
    |
    |-- PRO_E_COMP_PLACE_INTERFACE
    |
    |-- PRO_E_COMPONENT_SETS        (Placemet sets branch,
    |                                see ProAsmcomp.h)
    |-- PRO_E_COMPONENT_CONSTRAINTS (Placement constraints branch
    |                                see ProAsmcomp.h)
    |-- PRO_E_STD_FLEX_PROPAGATION
```

The following table describes the elements in the element tree for the Move and Move-Copy features:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Flex_Move.
PRO_E_FLEXMOVE_MOVED_GEOMETRY	Compound	Specifics the geometry to be moved.
PRO_E_FLEXMOVE_MOVED_COMPS	PRO_ELEM_TYPE_MULTI_VALUE	Optional element. This element is available only when you create the move feature at the assembly level. Specifies the collection of component references, that is, parts and subassemblies in an assembly, which are moved by the assembly level move feature.

Element ID	Data Type	Description
	Note The data type PRO_ELEM_TYPE_MULTI_VALUE enables you to assign multiple values to the element, though the data type is not an array.	
PRO_E_FLEXMOVE_DEFINE_METHOD	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the method to be used to move the entities. It takes the following values: • PRO_FLEXMOVE_DEF_METHOD_3D_DRAG—You can use the 3D transformation sets (in Creo Parametric interface using 3D Dragger) to define the flexible move feature. • PRO_FLEXMOVE_DEF_METHOD_DIMENSIONS—You can specify up to three constraining nonparallel linear dimensions or angular dimensions to define the flexible move feature. • PRO_FLEXMOVE_DEF_METHOD_CONSTRAINTS—You can specify component-like (assembly-like) constraints to define the flexible move feature.
PRO_E_D3ELEM_SETS	Array holder	Mandatory element when the definition method is PRO_FLEXMOVE_DEF_METHOD_3D_DRAG. An array holder of PRO_E_D3ELEM_SET elements.. The elements for 3D transformation sets are defined in ProD3Elem.h. For more information, see the section 3D Transformation Set Feature on page 1074.
PRO_E_FLEXMOVE_DIMS_COMPOUND	Compound	Mandatory element when the definition method is PRO_FLEXMOVE_DEF_METHOD_DIMENSIONS. Specifies the dimension references and arrays for the dimension definition method.

Element ID	Data Type	Description
PRO_E_FLEX_OPTS_CMPND	Compound	Mandatory element that contains the flexible modeling geometry attachment options to attach the moved surfaces. Specifies the integer and chain collection type elements. The elements related to reattachment of geometry in flexible modeling are defined in ProFlxmdlOpts.h. For more information, see the section Attachment Geometry Feature on page 1082.
PRO_E_FLXSLV_PROP_CONSTRS	Array	Optional element. Specifies an array that contains the tangency conditions and the reference geometry elements for tangency propagation. The elements related to propagation of tangency in flexible modeling are defined in ProFlexTanPropOpts.h. For more information, see the section Tangency Propagation on page 1099.
PRO_E_COMPONENT_SETS	Array holder	Mandatory element when the definition method is PRO_FLEXMOVE_DEF_METHOD_CONSTRAINTS. Specifies the constraint sets. The elements for constraint sets are defined in ProAsmcomp.h. For more information, see the section Constraint Sets and Mechanism Connections on page 1167 of chapter Assembly: Assembling Components on page 1159.

Element ID	Data Type	Description
PRO_E_COMPONENT_CONSTRAINTS	Array holder	Mandatory element when the definition method is PRO_FLEXMOVE_DEF_METHOD_CONSTRAINTS. Specifies the constraints. The elements for constraints are defined in ProAsmcomp.h. For more information, see the section Placement Constraints on page 1172 of chapter Assembly: Assembling Components on page 1159.
PRO_E_STD_FLEX_PROPAGATION	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies a pattern or mirror recognition feature to propagate the move feature changes.

Moved Geometry

The feature PRO_E_FLEXMOVE_MOVED_GEOMETRY is a compound element that allows you to select the geometries to be moved and the geometries to be excluded.

PRO_E_FLEXMOVE_MOVED_GEOMETRY

```
PRO_E_FLEXMOVE_MOVED_GEOMETRY
   |
   |-- PRO_E_STD_SURF_COLLECTION_APPL
   |
   |-- PRO_E_FLEXMOVE_EXCLUDED_GEOMETRY
   |       |
   |       |-- PRO_E_STD_SURF_COLLECTION_APPL
   |
   |-- PRO_E_FLEXMOVE_DTM_REF
```

The following table lists the contents of PRO_E_FLEXMOVE_MOVED_GEOMETRY element.

Element ID	Data Type	Description
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the collection of surface sets of the geometries to be moved. It may also include surface regions.
PRO_E_FLEXMOVE_EXCLUDED_GEOMETRY	Compound	Optional element to exclude surfaces.

Element ID	Data Type	Description
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the collection of surfaces from the move surfaces collector which should be excluded from the move operation.
PRO_E_FLEXMOVE_DTM_REF	PRO_ELEM_TYPE_MULTI_VALUE **Note** The data type PRO_ELEM_TYPE_MULTI_VALUE allows you to assign multiple values to the element, though the data type is not an array.	Optional element. Specifies the collection of datum entities that should be moved with the moved geometry.

Dimension Elements

The element PRO_E_FLEXMOVE_DIMS_COMPOUND is a compound element that allows you to specify the dimensions to move the geometries.

PRO_E_FLEXMOVE_DIMS_COMPOUND

```
PRO_E_FLEXMOVE_DIMS_COMPOUND
   |
   |-- PRO_E_FLEXMOVE_DIMS_ARRAY
          |
          |-- PRO_E_FLEXMOVE_DIM_COMPOUND
                 |
                 |-- PRO_E_FLEXMOVE_DIM_TYPE
                 |
                 |-- PRO_E_FLEXMOVE_DIM_REFS
                 |
                 |-- PRO_E_FLEXMOVE_DIM_VALUE
```

The following table lists the contents of PRO_E_FLEXMOVE_DIMS_COMPOUND element. All the elements are mandatory for the PRO_FLEXMOVE_DEF_METHOD_DIMENSIONS definition method type.

Element ID	Data Type	Description
PRO_E_FLEXMOVE_DIMS_ARRAY	Array holder	Mandatory element. The dimensions array can contain up to three dimensions of PRO_FLEXMOVE_ DIM_TYPE_LINEAR type or a single dimension of PRO_FLEXMOVE_DIM_TYPE_ANGULAR type.
PRO_E_FLEXMOVE_DIM_COMPOUND	Compound	Mandatory element. Specifies the constraining dimensions to move

Element ID	Data Type	Description
		the geometries.
PRO_E_FLEXMOVE_DIM_REFS	PRO_ELEM_TYPE_MULTI_VALUE	Mandatory element. Specifies two references for the given dimension. Out of the two selected references one must belong to the moved geometry and the other must belong to the geometry that is not affected by the move operation. The valid values for reference selections are as follows: • Surface—SEL_3D_SRF • Line—SEL_3D_EDG, SEL_3D_CURVE, SEL_3D_AXIS • Points—SEL_3D_VERT, SEL_3D_PNT, SEL_CURVE_END The valid combinations for reference selections are as follows: • Surface-Surface • Surface-Line • Surface-Point • Line-Line • Line-Point
PRO_E_FLEXMOVE_DIM_VALUE	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the value of the linear or angular dimension. The valid values are as follows: • Linear Dimensions—[-1e6, 1e6] • Angular Dimensions—[0, 360]

3D Transformation Set Feature

This section describes how to construct and access the element tree for 3D Transformation Set features. It also shows how to create, redefine, and access the properties of these features.

Introduction

The D3 element tree branch is used by features that use the 3D transformation steps, for example in the `PRO_FEAT_FLEXMOVE` feature type. See `ProFlexMove.h`. The 3D transformation steps are usually defined using the 3D Dragger in the Creo Parametric user interface.

Refer to the Creo Parametric Part Modeling Help for more information on placement and orientation references, and Degrees of Freedom.

The Element Tree for 3D Transformation Sets

The element tree for the 3D Transformation Set feature is documented in the header file `ProD3Elem.h`, and is shown in the following figure:

Element Tree for 3D Transformation Sets

```
PRO_E_D3ELEM_SETS
   |
   |-- PRO_E_D3ELEM_SET
          |
          |-- PRO_E_D3ELEM_PLACEMENT_REFERENCE
          |
          |-- PRO_E_D3ELEM_ORIENTATION_REFERENCE
          |
          |-- PRO_E_D3ELEM_LOCATION
          |
          |-- PRO_E_D3ELEM_MOVES
                 |
                 |-- PRO_E_D3ELEM_MOVE
                        |
                        |-- PRO_E_D3ELEM_MOVE_TYPE
                        |
                        |-- PRO_E_D3ELEM_MOVE_VALUE
```

The following table describes the elements in the element tree for the 3D Transformation Set feature:

Element ID	Data Type	Description
`PRO_E_D3ELEM_SETS`	`Array holder`	An array holder of `PRO_E_D3ELEM_SET` elements.
`PRO_E_D3ELEM_SET`	`Compound`	A compound element representing a single transformations set.
`PRO_E_D3ELEM_PLACEMENT_REFERENCE`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Specifies the placement reference for the 3D transformation sets. The valid values for placement reference selection are as follows:

Element ID	Data Type	Description
		• Surface—SEL_3D_SRF • Line—SEL_3D_EDG, SEL_3D_CURVE, SEL_3D_AXIS • Points—SEL_3D_VERT, SEL_3D_PNT, SEL_CURVE_END • Coordinate System—SEL_3D_CSYS **Note** In case of an empty value for the placement reference, the default placement reference will be used.
PRO_E_D3ELEM_ORIENTATION_REFERENCE	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies up to two references to define the orientation for the 3D transformation sets. The valid values for orientation reference selection are as follows: • Surface—SEL_3D_SRF • Line—SEL_3D_EDG, SEL_3D_CURVE, SEL_3D_AXIS • Coordinate System—SEL_3D_CSYS **Note** In case of an empty value for the placement reference, the default placement reference will be used.
PRO_E_D3ELEM_LOCATION	PRO_VALUE_TYPE_INT	Optional element. Specifies whether the transformation set moves with the geometry for every move. It takes the following values: • PRO_D3_LOCATION_MOVING—This is the default value. The coordinate system of the transformation set moves with each move made in the set. • PRO_D3_LOCATION_FIXED—This value can be set only if the placement reference and the orientation reference

Element ID	Data Type	Description
		are both selected to be the same coordinate system (SEL_3D_CSYS). When this value is set, the transformation set's coordinate system retains its position and orientation during the move, only the selected geometry moves. The move step is defined in reference to this fixed coordinate system.
PRO_E_D3ELEM_MOVES	Array holder	An array holder of PRO_E_D3ELEM_MOVE element.
PRO_E_D3ELEM_MOVE	Compound	A compound element which represents a single move in the given set

Element ID	Data Type	Description
PRO_E_D3ELEM_MOVE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the move type to be used to move the geometry This element takes the following values: • PRO_D3_MOVE_TYPE_SURF —Specifies that the move is made to a UV coordinate on a surface, from its initial UV placement point on that surface. **Note** This move type is relevant only if the move set is placed on a surface. To set and get the UV parameters use the functions: ○ ProFeatureD3eleUvGet() ○ ProFeatureD3eleUvSet() • PRO_D3_MOVE_TYPE_EDGE —Specifies that the move is made to coordinate (ratio parameter) on an edge, from its initial placement on that edge. The position is recorded as a length ratio on the edge. **Note** This move type is relevant only if the move set is placed on an edge. To set and get the position ratio use the functions:

Element ID	Data Type	Description
		○ ProFeatureD3elem-RatioGet() ○ ProFeatureD3elem-RatioSet() • PRO_D3_MOVE_TYPE_FREE —Specifies the move using a transformation matrix. It stores the final 3D location of the moved geometry. To compute the final position of the moved geometry, the saved 3D location is used. **Note** This move type is relevant only if the placement reference and the orientation reference elements have default (empty) values. To set and get the position delta matrix use the functions: ○ ProFeatureD3elem-MatrixGet() ○ ProFeatureD3elem-MatrixSet() • PRO_D3_MOVE_TYPE_FREETRF —Specifies the move using a transformation matrix, similar to PRO_D3_MOVE_TYPE_FREE. It stores the

Element ID	Data Type	Description
		transformation matrix. To compute the final position of the moved geometry, the saved transformation matrix is used. **Note** This move type is relevant only if the placement reference and the orientation reference elements have default (empty) values. • PRO_D3_MOVE_TYPE_XMOVE—Specifies the linear translation along the current coordinate system's X-vector. • PRO_D3_MOVE_TYPE_YMOVE—Specifies the linear translation along the current coordinate system's Y-vector. • PRO_D3_MOVE_TYPE_ZMOVE—Specifies the linear translation along the current coordinate system's Z-vector. • PRO_D3_MOVE_TYPE_XROTATE—Specifies the rotational angle along the current coordinate system's X-vector. • PRO_D3_MOVE_TYPE_YROTATE—Specifies the rotational angle along the current coordinate system's Y-vector. • PRO_D3_MOVE_TYPE_ZROTATE—Specifies the rotational angle along the current coordinate system's Z-vector.
PRO_E_D3ELEM_MOVE_VALUE	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the translation distance or the rotational angle for the move command.

Element ID	Data Type	Description
		The valid values are as follows: • Linear Move types—[-1e6, 1e6] • Angular Move types—[0, 360]

Functions for the PRO_E_D3ELEM_MOVE_VALUE element

This section describes the functions to be used to set the move type for the element `PRO_E_D3ELEM_MOVE_VALUE`.

Functions Introduced:

- **ProFeatureD3elemUvGet()**
- **ProFeatureD3elemUvSet()**
- **ProFeatureD3elemRatioGet()**
- **ProFeatureD3elemRatioSet()**
- **ProFeatureD3elemMatrixGet()**
- **ProFeatureD3elemMatrixSet()**

The functions `ProFeatureD3elemUvGet()` and `ProFeatureD3elemUvSet()` get and set the position UV parameter for the `PRO_E_D3ELEM_MOVE_VALUE` element on a surface or plane.

Note

The functions `ProFeatureD3elemUvGet()` and `ProFeatureD3elemUvSet()` are relevant only when the element `PRO_E_D3ELEM_MOVE_TYPE` has its move value as `PRO_D3_MOVE_TYPE_SRF`.

The functions `ProFeatureD3elemRatioGet()` and `ProFeatureD3elemRatioSet()` get and set the position ratio for the `PRO_E_D3ELEM_MOVE_VALUE` element on an edge or entity.

Note

The functions `ProFeatureD3elemRatioGet()` and `ProFeatureD3elemRatioSet()` are relevant only when the element `PRO_E_D3ELEM_MOVE_TYPE` has its move value as `PRO_D3_MOVE_TYPE_EDGE`.

The functions `ProFeatureD3elemMatrixGet()` and `ProFeatureD3elemMatrixSet()` get and set the position delta matrix for the `PRO_E_D3ELEM_MOVE_VALUE` element.

Note

The functions `ProFeatureD3elemMatrixGet()` and `ProFeatureD3elemMatrixSet()` are relevant only when the element `PRO_E_D3ELEM_MOVE_TYPE` has its move value as `PRO_D3_MOVE_TYPE_FREE`.

Attachment Geometry Feature

This section describes how to construct and access the element tree for reattaching geometry when a flexible modeling feature modifies or transforms a geometry selection. It also shows how to create, redefine, and access the properties of the attachment features.

Introduction

When a geometry selection is transformed by a flexible modeling feature, you can reattach the geometry to the model in one of the following ways:

- Extend the surfaces of the geometry selection and the neighboring geometry until they intersect each other, or until the hole where the geometry selection was originally located is closed and the transformed or modified geometry is reattached.
- Create the side surfaces to close the gap between the moved geometry and the hole left in the model.
- Recreate immediate neighboring surfaces maintaining tangency conditions, if the moved geometry and the immediate neighboring surfaces are tangent planes and circles.

The attachment element `PRO_E_FLEX_OPTS_CMPND` is seen in the following feature types:

- `PRO_FEAT_FLEXMOVE` (See `ProFlexMove.h`)
- `PRO_FEAT_FLX_OGF` (See `ProFlexOffset.h`)
- `PRO_FEAT_ANALYT_GEOM` (See `ProFlexMag.h`)
- `PRO_FEAT_FLXATTACH` (See `ProFlexAttach.h`)
- `PRO_FEAT_FLEXSUBST` (See `ProFlexSubstitute.h`)

The Element Tree for Attachment Geometry Options

The element tree for the Attachment Geometry feature is documented in the header file `ProFlxmdlOpts.h`, and is shown in the following figure.

Element Tree for Attachment Geometry Options

```
PRO_E_FLEX_OPTS_CMPND
    |
    |-- PRO_E_FLEX_TRF_SEL_ATT_GEOM
    |
    |-- PRO_E_FLEX_ATTACH_GEOM
    |
    |-- PRO_E_FLEX_CR_RND_GEOM
    |
    |-- PRO_E_FLEX_KEEP_ORIG_GEOM
    |
    |-- PRO_E_FLEX_PROPAGATE_TANGENCY
    |
    |-- PRO_E_FLEX_DFLT_CONDITIONS
    |
    |-- PRO_E_FLEX_BOUND_EDGES_CMP
    |       |
    |       |-- PRO_E_STD_CURVE_COLLECTION_APPL
    |
    |-- PRO_E_FLEX_SOL_INDEX
    |
    |-- PRO_E_FLEX_MAINTAIN_TOPO
    |
    |-- PRO_E_FLEX_PULL_OPTION
    |
    |-- PRO_E_FLEX_ATT_CHNS_CMP
    |
    |-- PRO_E_FLEX_SPLIT_EXT_SURFS_CMP
```

The following table describes the elements in the element tree for the Geometry Attachment feature:

Element ID	Data Type	Description
PRO_E_FLEX_OPTS_CMPND	Compound	Specifies the flexible modeling geometry reattachment options.
PRO_E_FLEX_TRF_SEL_ATT_GEOM	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies whether the chamfers and rounds that attach the moved geometry to the model must also be transformed.

Element ID	Data Type	Description
		This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF, PRO_FEAT_FLEXSUBST, mirror feature and in patterning. This element takes the following values: • PRO_FLXMDL_OPT_YES • PRO_FLXMDL_OPT_NO When the value PRO_FLXMDL_OPT_NO is specified, the chamfers and rounds are removed and optionally recreated.
PRO_E_FLEX_ATTACH_GEOM	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies whether to attach the moved geometry to the same quilt or solid it was detached from. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM. This element takes the following values: • PRO_FLEXMODEL_OPT_YES—Attaches the moved geometry to same quilt or solid. • PRO_FLEXMODEL_OPT_NO—Creates a new separate quilt.
PRO_E_FLEX_CR_RND_GEOM	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies whether to create a round geometry after the geometry selection is moved and reattached. This applicable for geometry selection that was originally attached by round geometry. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF, PRO_FEAT_ANALYT_GEOM and PRO_FEAT_FLEXSUBST.

Element ID	Data Type	Description
		This element takes the following values: • PRO_FLEXMODEL_OPT_YES • PRO_FLEXMODEL_OPT_NO There are two cases here: • When PRO_E_FLEX_ATTACH_GEOM is set to PRO_FLEXMODEL_OPT_YES and PRO_E_FLEX_CR_RND_GEOM is set to following values: ○ PRO_FLEXMODEL_OPT_YES—The rounds are recreated. ○ PRO_FLEXMODEL_OPT_NO—The rounds are not recreated. • When PRO_E_FLEX_ATTACH_GEOM is set to PRO_FLEXMODEL_OPT_No the rounds are not reattached. However, in this case you can save the information about the attachment properties for the rounds. PRO_E_FLEX_CR_RND_GEOM is set to following values: ○ PRO_FLEXMODEL_OPT_YES—The attachment information of the rounds is stored in intent objects. This information can be used by some other features. ○ PRO_FLEXMODEL_OPT_NO—The attachment information of the rounds is not stored.
PRO_E_FLEX_KEEP_ORIG_GEOM	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies if the original geometry must be moved or a copy of the geometry should be moved.

Element ID	Data Type	Description
		This element is relevant in PRO_FEAT_FLEXMOVE and PRO_FEAT_FLEXSUBST. This element takes the following values: • PRO_FLEXMODEL_OPT_YES • PRO_FLEXMODEL_OPT_NO
PRO_E_FLEX_PROPAGATE_TANGENCY	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies if tangency must be maintained between the modified geometry and the neighboring geometry. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM. This element takes the following values: • PRO_FLEXMODEL_OPT_YES • PRO_FLEXMODEL_OPT_NO
PRO_E_FLEX_DFLT_CONDITIONS	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies if the default condition must be applied to the vertices of the dragged geometry. The vertices are selected by the system. The default condition is to fix the selected vertices. These vertices do not transform along with the dragged geometry. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM. This element takes the following values: • PRO_FLEXMODEL_OPT_YES • PRO_FLEXMODEL_OPT_NO
PRO_E_FLEX_BOUND_EDGES_CMP	Compound	This compound element collects the bounding edges.
PRO_E_FLEX_MAINTAIN_TOPO	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies the option to maintain solution topology for generic flexible

Element ID	Data Type	Description
		modeling features: • PRO_FEAT_FLEXMOVE • PRO_FEAT_FLX_OGF • PRO_FEAT_ANALYT_GEOM • PRO_FEAT_FLEXSUBST • PRO_FEAT_FLEXATTACH This element has the following valid values: • PRO_FLXMDL_OPT_YES— The application generates feature geometry that is topologically similar to the solution stored in the element PRO_E_FLEX_SOL_INDEX. If such a solution cannot be created, the feature will fail. • PRO_FLXMDL_OPT_NO— The application generates geometry that is topologically similar to the solution stored in the element PRO_E_FLEX_SOL_INDEX. If such a solution cannot be created, the application tries to create other successful solution. If the application cannot create any other solution for the changed model, the feature fails. **Note** You cannot select the solution interactively using Creo Parametric TOOLKIT applications.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain	Optional element. Collects the bounding edges from the geometry list to which the primary feature geometry will be reattached. The bounding edges are used as the limiting edges for the feature reattachment solutions. The edges that belong to the primary feature references cannot be used as bounding edges.

Element ID	Data Type	Description
		Note The information on the chosen solution is stored in the element PRO_E_FLEX_SOL_INDEX. This element is not accessible by Creo Parametric TOOLKIT, under which a default solution will be always used. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF, PRO_FEAT_ANALYT_GEOM, PRO_FEAT_FLEXSUBST and PRO_FEAT_FLEXATTACH.
PRO_E_FLEX_PULL_OPTION	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies the attachment option by which the moved geometry will be attached to the model. The bounding chain edges selected in the element PRO_E_FLEX_BOUND_EDGES_CMP are used for the reattachment of moved geometry. This element takes the following values: • PRO_FLEXMODEL_PULL_NONE—The moved geometry is attached using the default method or according to the contents of the **Create side surfaces** and **Extend side surfaces** chain collectors which decide the attachment method. Refer to the section Default Method for the Pull Option on page 1092 for more information on the default method. • PRO_FLEXMODEL_PULL_CREATE_SURFS—The moved geometry is reattached to the same quilt or solid it was

Element ID	Data Type	Description
		detached from using the **Create side surfaces** option. The moved surfaces are attached to the model by creating surfaces that connect the original contour to the final contour of the moved surface. • PRO_FLEXMODEL_PULL_EXTEND_SURFS—The moved geometry is reattached to the same quilt or solid it was detached from using the **Extend side surfaces** option. The moved surfaces are attached to the model by extending and intersecting the moved surfaces and their neighboring surfaces. This element is relevant in PRO_FEAT_FLEXMOVE.
PRO_E_FLEX_ATT_CHNS_CMP	Compound	Non-default edge chain collector.
PRO_E_FLEX_SIDE_SRFS_CMP	Compound	Edge chain collector for **Create side surfaces**.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Optional element. Collects the edge chains for which the attachment option will be changed from default to **Create side surfaces**. Here side surfaces will be created to close the gap between the moved geometry and the hole left in the model. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM. You can select the edges and intent edges that belong to the intersection between the geometry selection and the neighboring geometry.
PRO_E_FLEX_EXT_INT_CMP	Compound	Edge chain collector for **Extend and intersect**.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Optional element. Collects the edge chains for which the attachment option will be changed

Element ID	Data Type	Description
		from default to **Extend and intersect**. Here the surfaces of the geometry selection and the neighboring geometry are extended until they intersect each other. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM. You can select the edges and intent edge that belong to the intersection between the geometry selection and the neighboring geometry, and do not belong to the side surface set.
PRO_E_FLEX_SPLIT_EXT_SURFS_CMP	Compound	Collector to split the extending surface area. This element is relevant in PRO_FEAT_FLEXMOVE and PRO_FEAT_FLX_OGF.
PRO_E_FLEX_EXT_SRFS_CMP	Compound	Compound collector for extending surfaces set to be split.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Optional element. This element is used with the splitting surfaces set and the flip options. Collects the extending surfaces that will be split by the splitting surfaces. You can select the surfaces that belong to the flexible modeling feature's references surface set. This element is relevant in PRO_FEAT_FLEXMOVE and PRO_FEAT_FLX_OGF. This element is irrelevant in PRO_FEAT_ANALYT_GEOM as the reference surface is the only possible extending surface.
PRO_E_FLEX_SPT_SRFS_CMP	Compound	Compound collector for splitting surfaces set.

Element ID	Data Type	Description
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Optional element. This element is used with the extending surfaces set. Collects surfaces that will be extended and used to split the extending surfaces. You can select the surfaces which belong to the same solid or quilt that contains the surfaces being modified. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM.
PRO_E_FLEX_FLIP_SPLIT_DIR	PRO_VALUE_TYPE_INT	Optional element. This element is used with the extending and splitting surfaces sets. Flips the side of the extending surfaces. This element is relevant in PRO_FEAT_FLEXMOVE, PRO_FEAT_FLX_OGF and PRO_FEAT_ANALYT_GEOM. This element takes the following values: • PRO_FLEXMODEL_OPT_YES • PRO_FLEXMODEL_OPT_NO

Default Method for the Pull Option

When the value of the element PRO_E_FLEX_PULL_OPTION is set to PRO_FLEXMODEL_PULL_NONE the moved geometry is attached using either of the following options:

- Default method
- According to the contents of the **Create side surfaces** and **Extend side surfaces** chain collectors

This section describes the default method for the Pull option in detail.

The geometry of the modified surface and the neighboring surface decides the default method by which the surfaces will be attached:

- If the modified surface and the neighboring surface are not tangential to each other, the surfaces are extended until they intersect, that is, they are attached using the **Extend and intersect** option.
- If the modified surface and the neighboring surface are tangential to each other, the moved geometry is attached in one of the following ways:
  - If the modified surface is of any type and the neighboring surface is a round surface, the surfaces are attached in the following way:
    - The round is removed.
    - The surfaces are attached using the **Extend and intersect** option.
    - The round is recreated.
  - If the modified surface is of any type and the neighboring surface is not a round surface, the surfaces are attached in either of the following ways:
    - The surfaces are attached by creating side surfaces, that is, by using the **Create side surfaces** option.
    - If the side wall construction fails due to some geometrical reason, for example, self-intersection, then the default option is reset to **Extend and intersect**.
  - If modified surface and the neighboring surface are analytic, that is, cylindrical, planar, conical, toroidal, or tabulated cylinder and the tangency propagation is set to No, the surfaces are attached in either of the following ways:
    - If the neighboring surface is identified as a round, then the attachment is similar to the above mentioned round surface.
    - If the neighboring surface is not a round surface, then the surfaces are attached using the **Create side surfaces** option.
  - If modified surface and the neighboring surface are analytic, that is, cylindrical, planar, conical, toroidal, or tabulated cylinder, the tangency is along an isoline and the tangency propagation is set to Yes, the surfaces are attached in either of the following ways:
    - If the neighboring surface is identified as a round, then the attachment is similar to the above mentioned round surface.
    - If the neighboring surface is not a round surface, the neighboring surface is dragged to maintain the tangency between the surfaces.

      Refer to the section Tangency Propagation on page 1099 for more information on tangency propagation and dragged geometry.

Offset Geometry Feature

This section describes how to construct and access the element tree for Offset Geometry feature. It also shows how to create, redefine, and access the properties of this feature.

Introduction

The Offset Geometry feature allow you to offset a geometry selection that belongs to a solid geometry or to a quilt, and reattach it back to the solid or quilt. You can offset the following geometry selection:

- Any surface collection.
- An intent surface.
- Any combination of the above geometries.

A offset feature will act on a single geometry selection. To offset a different geometry selection, a new Offset Geometry feature must be created.

The Element Tree for Offset Geometry

The element tree for the Offset Geometry feature is documented in the header file `ProFlexOffset.h`, and is shown in the following figure.

Element Tree for Offset Geometry

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_STD_SURF_COLLECTION_APPL
    |
    |-- PRO_E_OGF_OFFSET_VAL
    |
    |-- PRO_E_OGF_DIR_OPT
    |
    |-- PRO_E_FLEX_OPTS_CMPND  (General FLX-MDL options branch,
    |                           see ProFlxmdlOpts.h)
    |
    |-- PRO_E_FLXSLV_PROP_CONSTRS (General FLX-MDL tangency
    |                              propagation branch,
    |                              see ProFlexTanPropOpts.h)
    |-- PRO_E_STD_FLEX_PROPAGATION
```

The following table describes the elements in the element tree for the Offset Geometry feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Offset_Geom.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the collection of surfaces to offset.
PRO_E_OGF_OFFSET_VAL	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the offset value. It takes value between [-1.00e+06, 1.00e+06].
PRO_E_OGF_DIR_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of offset. It takes the following values: • PRO_FLXOGF_DIR_NORMAL—Offsets the geometry normal to the selected surface. • PRO_FLXOGF_DIR_FLIP—Flips the direction of offset.
PRO_E_FLEX_OPTS_CMPND	Compound	Mandatory element that contains the flexible modeling geometry attachment options to attach the surfaces offset. Specifies the integer and chain collection type elements. The elements related to reattachment of geometry in flexible modeling are defined in ProFlxmdlOpts.h. For more information, see the section Attachment Geometry Feature on page 1082.

Element ID	Data Type	Description
PRO_E_FLXSLV_PROP_CONSTRS	Array	Optional element. Specifies an array that contains the tangency conditions and the reference geometry elements for tangency propagation. This element must be specified when the element PRO_E_FLEX_PROPAGATE_TANGENCY is set to PRO_FLEXMODEL_OPT_YES in the header file ProFlxmdlOpts.h. The elements related to propagation of tangency in flexible modeling are defined in ProFlexTanPropOpts.h. For more information, see the section Tangency Propagation on page 1099.
PRO_E_STD_FLEX_PROPAGATION	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies a pattern or mirror recognition feature to propagate the offset geometry feature changes.

Modify Analytic Surface Feature

This section describes how to construct and access the element tree for Modify Analytic Surface feature for flexible modeling. It also shows how to create, redefine, and access the properties of this feature.

Introduction

The Modify Analytic Surface feature allows you to modify the analytic surfaces. The modification is done by creating and modifying basic dimensions that define each surface type.

The following analytic surfaces can be modified:

- Cylinder—The axis remains fixed and the radius can be modified.
- Torus—The axis of revolution of the circle remains fixed. The radius of the circle and the distance (radius) from the center of the circle to the axis of revolution can be modified.
- Cone—The axis and vertex of the cone remain fixed, and the angle can be modified.

The Element Tree for Modify Analytic Surface

The element tree for the Modify Analytic Surface feature is documented in the header file `ProFlexMag.h`, and is shown in the following figure.

Element Tree for Modify Analytic Surface

```
PRO_E_FEATURE_TREE
   |
   |-- PRO_E_FEATURE_TYPE
   |
   |-- PRO_E_STD_FEATURE_NAME
   |
   |-- PRO_E_STD_SURF_COLLECTION_APPL
   |
   |-- PRO_E_STD_FLEX_PROPAGATION
   |
   |-- PRO_E_FLEX_OPTS_CMPND    (General FLX-MDL options branch,
   |                             see ProFlxmdlOpts.h)
   |-- PRO_E_FLXSLV_PROP_CONSTRS (General FLX-MDL tangency
   |                              propagation branch,
   |                              see ProFlexTanPropOpts.h)
   |-- PRO_E_MAG_ANGLE_VAL      (Conic surface)
   |
   |-- PRO_E_MAG_RADII_VAL      (Cylindric and toroidal surface)
   |
   |-- PRO_E_MAG_RADII2_VAL     (Toroidal surface)
```

The following table describes the elements in the element tree for the Modify Analytic Surface feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is MODIFY_ANALYTIC_SURFACE.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the collection of analytical surface sets to be modified. The valid surface selections are: cylindrical, conical or toroidal surfaces. **Note** You can specify only one reference surface or surface region at a time in this element.
PRO_E_STD_FLEX_PROPAGATION	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies a pattern or mirror recognition feature to propagate the modified analytic surface feature changes.
PRO_E_FLEX_OPTS_CMPND	Compound	Mandatory element that contains the flexible modeling geometry

Element ID	Data Type	Description
		attachment options to attach the modified surfaces. Specifies the integer and chain collection type elements. The elements related to reattac of geometry in flexible modeling are defined in `ProFlxmdlOpts.h`. For more information, see the section Attachment Geometry Feature on page 1082.
`PRO_E_FLXSLV_PROP_CONSTRS`	`Array`	Optional element. Specifies an array that contains the tangency conditions and the reference geometry elements for tangency propagation. This element must be specified when the element `PRO_E_FLEX_PROPAGATE_TANGENCY` is set to `PRO_FLEXMODEL_OPT_YES` in the header file `ProFlxmdlOpts.h`. The elements related to propagation of tangency in flexible modeling are defined in `ProFlexTanPropOpts.h`. For more information, see the section Tangency Propagation on page 1099.
`PRO_E_MAG_ANGLE_VAL`	`PRO_VALUE_TYPE_DOUBLE`	Mandatory element for conic surface. Specifies the angle of the conic surface. The element takes angular value between [0.5, 89] degrees.

Element ID	Data Type	Description
PRO_E_MAG_RADII_VAL	PRO_VALUE_TYPE_DOUBLE	Mandatory element for cylindrical and a toroidal surface. For cylindrical surface, the element specifies the radius of the cylindrical surface. For torodial surface, the element specifies the radius of revolution of the toroidal surface. The element takes value between [0, 1.00e+06].
PRO_E_MAG_RADII2_VAL	PRO_VALUE_TYPE_DOUBLE	Mandatory element for conic surface. Specifies the circle radius of the toroidal surface. The element takes value between [0, 1.00e+06].

Tangency Propagation

This section describes how to construct and access the elements for propagating tangency in a flexible modeling feature.

Introduction

When geometry is modified with **Move**, **Offset**, or **Modify Analytic** command, you can maintain the tangency between the modified geometry and the neighboring geometry. To maintain the tangency, the neighboring geometry may be modified, though it was not selected for modification. The geometry that is included during tangency propagation is defined as dragged geometry. The dragged geometry is modified in such a way that it always remains tangential to the directly modified geometry. The tangency propagation stops when the application recognizes a surface that is round or chamfer. The chamfer or round surface, except variable rounds, is recreated after the geometry is modified. The round and chamfer geometry are called connecting geometry.

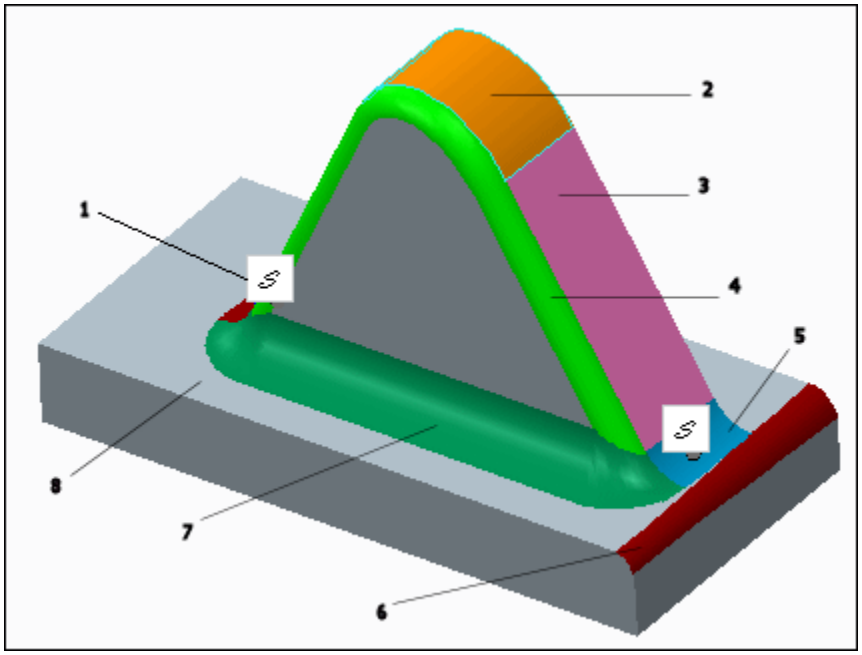
Depending on the type of round surface, the tangency may or may not be propagated. The rounds are of the following types:

- Rounds that can be propagated. Tangency propagation can be forced to be carried through and continue onto adjacent tangent geometry.
- Rounds that cannot be propagated. The tangency stops and cannot be forced to go further.

- Rounds that cannot be recreated. These are mainly the variable rounds. Variable rounds can be removed but cannot be recreated.
- Interfering rounds. These are rounds that do not connect transformed or dragged geometry to the rest of the model but have to be removed and recreated to accommodate the changes in the transformed geometry, dragged geometry, and other rounds.

The round or chamfer surfaces connect the directly modified geometry and the dragged surfaces to the background geometry. The background geometry is the base geometry that is not modified.

The modified, dragged, connecting, fixed, and background geometry are displayed in different colors during modification in the Creo Parametric user interface as shown in the figure below:



1—Default vertices are automatically created when you propagate tangency

2—Directly modified geometry

3—Dragged geometry

4—Rounds that cannot propagated

5—Rounds that can be propagated

6—Rounds that cannot be recreated

7—Interfering rounds

8—Background geometry

You can control the changes in geometry during tangency propagation by specifying the tangency constraints. During tangency propagation the fixed vertex constraints are also considered. The tangency constraint is applied on a reference geometry that can either be the dragged geometry or the connecting geometry, or both. Refer to Creo Flexible Modeling Help for more information.

To work with tangency propagation, you must set the value of the element `PRO_E_FLEX_PROPAGATE_TANGENCY` to `PRO_FLEXMODEL_OPT_YES`. The element is defined in the header file `ProFlxmdlOpts.h`.

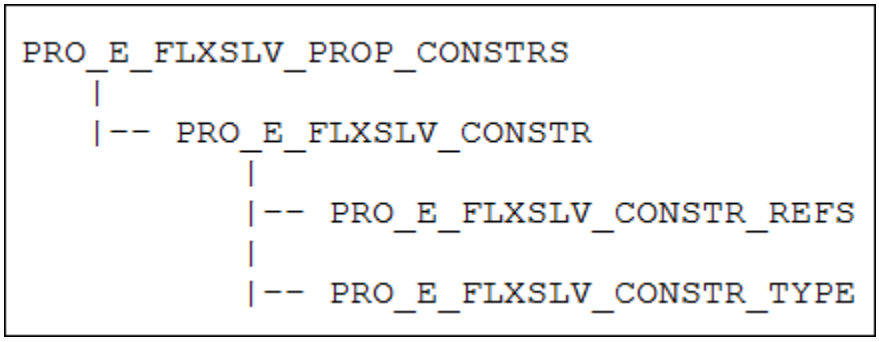
The element `PRO_E_FLXSLV_PROP_CONSTRS` is used to set the conditions that controls the changes in geometry during tangency propagation. It is seen in the following feature types:

- `PRO_FEAT_FLEXMOVE` (See `ProFlexMove.h`)
- `PRO_FEAT_FLX_OGF` (See `ProFlexOffset.h`)
- `PRO_FEAT_ANALYT_GEOM` (See `ProFlexMag.h`)

The Element Tree for Tangency Propagation

The element tree for the Tangency Propagation is documented in the header file `ProFlexTanPropOpts.h`, and is shown in the following figure.

Element Tree for Tangency Propagation



The following table describes the elements in the element tree for the Tangency Propagation:

Element ID	Data Type	Description
`PRO_E_FLXSLV_PROP_CONSTRS`	`Array`	Specifies an array of conditions that control the tangency propagation.
`PRO_E_FLXSLV_CONSTR`	`Compound`	Mandatory element. Specifies a single condition for tangency along with the reference geometry.

Element ID	Data Type	Description
PRO_E_FLXSLV_CONSTR_REFS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the selection of reference geometry.
PRO_E_FLXSLV_CONSTR_TYPE	PRO_VALUE_TYPE_INTEGER	Mandatory element. Specifies a condition that sets constraint on the neighboring surfaces during the propagation of tangency. The valid values are defined in the enumerated data type ProFlxmdlPropOptFlag. Refer to section Setting Conditions for Tangency Propagation on page 1102 for more information on tangency conditions.

Setting Conditions for Tangency Propagation

You can set the conditions depending on geometry type that must be considered during tangency propagation. The conditions are specified using the enumerated data type ProFlxmdlPropOptFlag. The valid values are:

- PRO_FLXSLV_CONSTR_TYPE_FIXED—Specifies that the reference geometry must be fixed. Tangency is not maintained between the modified geometry and the neighboring geometry. You can set the value for following types of geometry:
  - Planar surface
  - Cylindrical surface
  - Conical surface
  - Toroidal surface
  - Spherical surface
  - Tabulated cylinder
  - Surface of revolution
  - Round surface
  - Edge (surface isoline) on the surface
  - Vertex on the surface

Note

You can specify the condition PRO_FLXSLV_CONSTR_TYPE_FIXED while manipulating the geometry of parts, using the Creo Flexible Modeling commands, even if the value of PRO_E_FLEX_PROPAGATE_TANGENCY is set to PRO_FLEXMODEL_OPT_NO. Here the reference round and chamfer geometry are not considered as rounds and chamfers.

- PRO_FLXSLV_CONSTR_TYPE_FIX_AXIS—Specifies that the axis of the reference geometry must be fixed. You can set the value for following types of geometry:
  - Cylindrical surface
  - Conical surface
  - Toroidal surface
  - Spherical surface
  - Surface of revolution
- PRO_FLXSLV_CONSTR_TYPE_FIX_CNTR—Specifies that the center of the reference geometry must be fixed. You can set the value for following types of geometry:
  - Toroidal surface
  - Spherical surface
- PRO_FLXSLV_CONSTR_TYPE_FIX_NORM—Specifies that the reference geometry must be normal to the directly modified geometry. You can set the value for following type of geometry:
  - Planar surface
- PRO_FLXSLV_CONSTR_TYPE_CONST_R1—Specifies that the minor radius of reference geometry must be constant. You can set the value for following types of geometry:
  - Cylindrical surface
  - Toroidal surface—The minor diameter is kept constant.
  - Spherical surface
- PRO_FLXSLV_CONSTR_TYPE_CONST_R2—Specifies that the major radius of reference geometry must be constant. You can set the value for following type of geometry:
  - Toroidal surface

- PRO_FLXSLV_CONSTR_TYPE_CONST_ANG—Specifies that the angle between the reference geometry and directly modified geometry must be constant. You can set the value for following type of geometry:
  - Conical surface
- PRO_FLXSLV_CONSTR_TYPE_FIX_POLE—Specifies that the position of the pole in the reference geometry must be fixed. You can set the value for following type of geometry:
  - Conical surface
- PRO_FLXSLV_CONSTR_TYPE_PRPG_THRU—Specifies that the tangency must be propagated till the last available round surface or till the surface after which the tangency will break.
  - Round surface
  - Cylindrical surface
- PRO_FLXSLV_CONSTR_TYPE_FIX_RNDEDG_PNT—Specifies that the endpoint (vertex) on the edge of the reference geometry must be fixed. You can set the value for following type of geometry:
  - Round surface
- PRO_FLXSLV_CONSTR_FIX_WITH_TNGCY—Specifies that the tangency must be maintained between the modified geometry and the neighboring dragged geometry, and further between the neighboring dragged geometry and connecting geometry. You can set the value for following types of geometry:
  - Planar surface
  - Cylindrical surface
  - Conical surface
  - Toroidal surface
  - Spherical surface
  - Tabulated cylinder
  - Surface of revolution
  - Round surface
  - Edge (surface isoline) on the surface
  - Vertex on the surface
- PRO_FLXSLV_CONSTR_TYPE_CONST_R—Specifies that the radius of reference geometry must be constant. You can set the value for following types of geometry:
  - Cylindrical surface

- Spherical surface

- `PRO_FLXSLV_CONSTR_TYPE_KEEP_SPHERICAL`—Specifies that the shape of the sphere does not change, though the radius can change. You can set the value for following types of geometry:
  - Spherical surface

Mirror Feature

This section describes how to construct and access the element tree for Mirror feature for flexible modeling. It also shows how to create, redefine, and access the properties of this feature.

Introduction

The Mirror feature allows you to mirror a selected set of geometry about a plane. You can either attach the mirrored geometry to the solid or quilt from which it was created or keep it detached.

If the geometry selection for the mirror feature includes or is attached by round geometry, then the round geometry can be recreated in the new mirrored location.

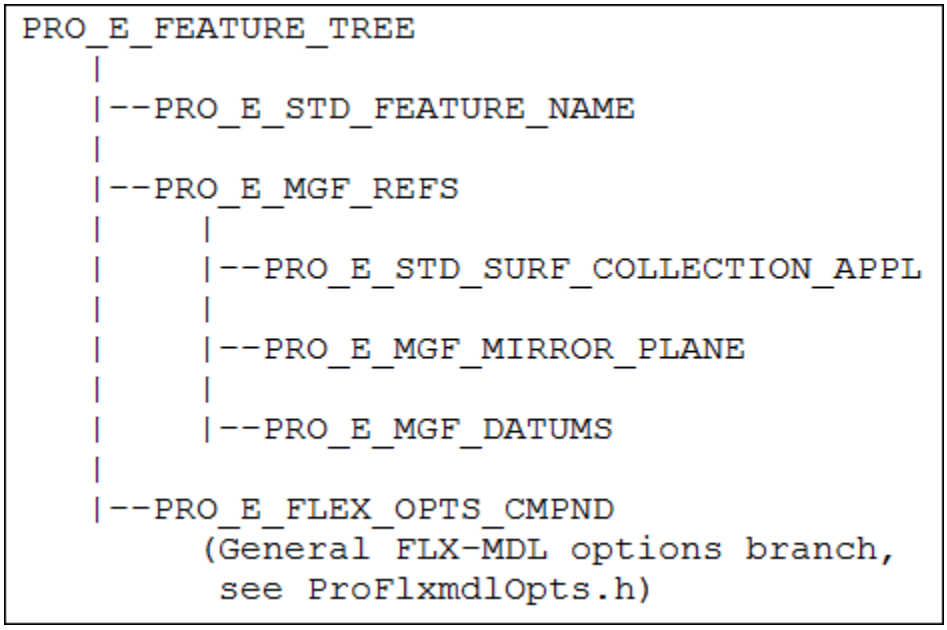
Note

The group header of the Mirror feature behaves as a standard feature. You can extract the header element tree in Creo Parametric TOOLKIT.

The Element Tree for Mirror

The element tree for the Mirror feature is documented in the header file `ProFlexMirror.h`, and is as shown in the following figure:

Element Tree for Mirror



The following table describes the elements in the element tree for the Mirror feature:

Element ID	Data Type	Description
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Mirror_Geometry.
PRO_E_MGF_REFS	Compound	Compound element that specifics the geometry, curves, and datums to be mirrored.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the collection of surface sets of the geometry to be mirrored.
PRO_E_MGF_MIRROR_PLANE	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum plane or intent datum plane about which the geometry will be mirrored.

Element ID	Data Type	Description
PRO_E_MGF_DATUMS	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the collection of curves and datum entities to be mirrored.
PRO_E_FLEX_OPTS_CMPND	Compound	Mandatory element that contains the Creo Flexible Modeling geometry attachment options to attach the mirrored surfaces. Specifies the integer and chain collection type elements. The elements related to reattachment of geometry in flexible modeling are defined in ProFlxmdlOpts.h. For more information, see the section Attachment Geometry Feature on page 1082.

Substitute Feature

This section describes how to construct and access the element tree for Substitute feature. It also shows how to create, redefine, and access the properties of this feature.

Introduction

The Substitute feature allows you to replace a geometry selection that belongs to a solid geometry or a quilt with replacing surfaces. The replacing surfaces are attached to the solid or quilt, and any round geometry between the geometry selection. The model is recreated after the replacing geometry is attached.

You can substitute the following geometry selection:

- Any surface collection.
- An intent surface.
- Any combination of the above geometries.
- Any one-sided edges on surfaces or quilts.

All the surfaces and one-sided edges in the geometry selection must belong to the same solid geometry or to the same quilt. The geometry selection must not be tangential to the neighboring geometry, or should be attached to the neighboring geometry with round geometry.

The replacing surfaces must be large enough to attach to the extension of the neighboring geometry, without the need to extend the replacing surfaces.

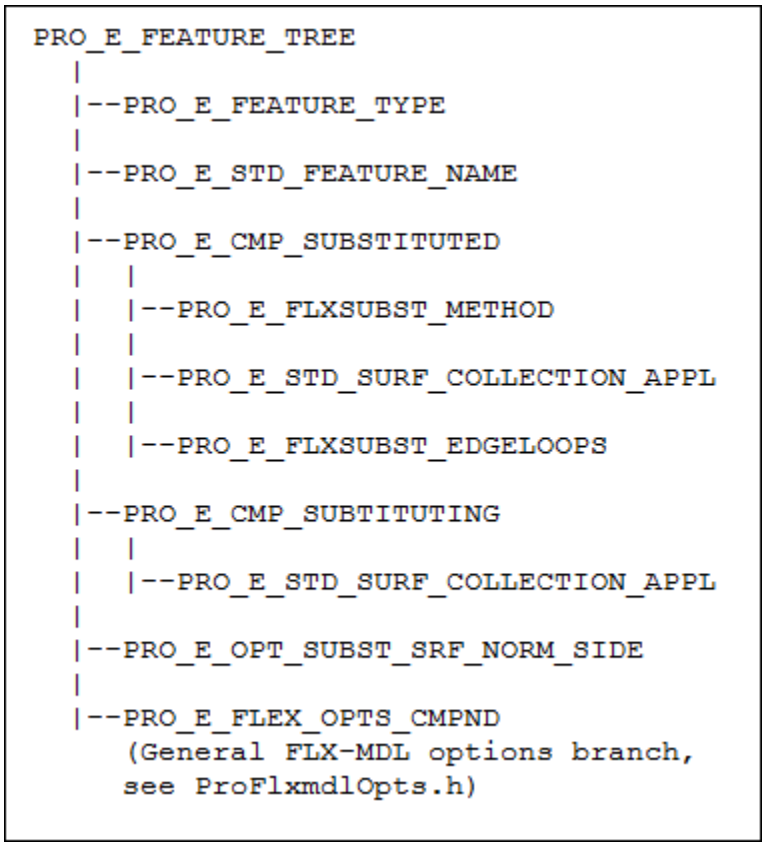
After substitution, the replacing surfaces that define the one-sided edge loops are extended and trimmed so that the resulting edges lie on the substituting geometry.

Refer to the Creo Flexible Modeling Help for more information.

The Element Tree for Substitute

The element tree for the Substitute feature is documented in the header file `ProFlexSubstitute.h`, and is shown in the following figure.

Element Tree for Substitute



The following table describes the elements in the element tree for the Substitute feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of feature. The value of this feature must be PRO_FEAT_FLEXSUBST.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Substitute.
PRO_E_CMP_SUBSTITUTED	Compound	Compound element for surfaces to be substituted.
PRO_E_FLXSUBST_METHOD	PRO_VALUE_TYPE_INT	Specifies which kind of geometry must be substituted. The valid geometry types are:

Element ID	Data Type	Description
		• PRO_FLEXSUBST_SURFACES—Specifies that surfaces must be substituted. • PRO_FLEXSUBST_LOOPS—Specifies that one-sided edges must be substituted.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element for PRO_FLEXSUBST_SURFACES. Specifies the collection of surfaces that will be replaced by the substituting surfaces.
PRO_E_FLXSUBST_EDGELOOPS	PRO_VALUE_TYPE_SELECTION	Mandatory element for PRO_FLEXSUBST_LOOPS. Specifies the collection of one-sided edges that will be replaced by the substituting surfaces.
PRO_E_CMP_SUBTITUTING	Compound	Compound element for substituting surfaces.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the collection of substituting surfaces.
PRO_E_OPT_SUBST_SRF_NORM_SIDE	PRO_VALUE_TYPE_BOOLEAN	Mandatory element. Specifies the direction of the normal vectors of the substituting surfaces. This element takes the following values: • PRO_B_TRUE—Specifies that the side 1 is direction of normal vector. • PRO_B_FALSE—Specifies that the side 2 is direction of normal vector.
PRO_E_FLEX_OPTS_CMPND	Compound	Mandatory element that contains the flexible modeling geometry attachment options to attach the substituting surfaces. Specifies the integer and chain collection type elements. The elements related to reattachment of geometry in flexible modeling are defined in ProFlxmdlOpts.h. For more information, see the section Attachment Geometry Feature on page 1082.

Planar Symmetry Recognition Feature

This section describes how to construct and access the element tree for Planar Symmetry Recognition feature. It also shows how to create, redefine, and access the properties of this feature.

Introduction

The Planar Symmetry Recognition feature allows you to identify and modify geometry that is symmetric with respect to a plane. The modifications made to the geometry on one side are automatically propagated to the other side and the symmetry is kept.

The Planar Symmetry Recognition feature identifies symmetrical geometry in the following ways:

- You can collect two seed surfaces or surface regions that are symmetric. The feature computes the plane of symmetry, and finds all pairs of neighboring surfaces and surface regions which are symmetric with respect to the symmetry plane. The propagation ends when non-symmetric neighboring surfaces are found.

  The two seed surfaces or regions must belong to:

  - The solid geometry.
  - A single quilt.
  - Two quilts.
- You can collect a seed surface or surface region and a plane of symmetry. The feature finds the mirror image of the seed surface or surface region and, finds all the pairs of neighboring surfaces and surface regions which are symmetric with respect to the symmetry plane.

According to the correspondence between the geometry of the symmetrical members, the following variation of planar symmetry recognition are possible:

- **Identical**—There is exact correspondence between the surfaces of the symmetrical members as well as between the intersection edges defined by these members and the surrounding geometry.
- **Similar**—There is exact correspondence between the surfaces of the symmetrical members, but there is no exact correspondence between the intersection edges defined by these members and the surrounding geometry. The number of intersection loops must be the same, but the type of edges, number of edges in each intersection loop and the intersected model surfaces do not have to be the same.

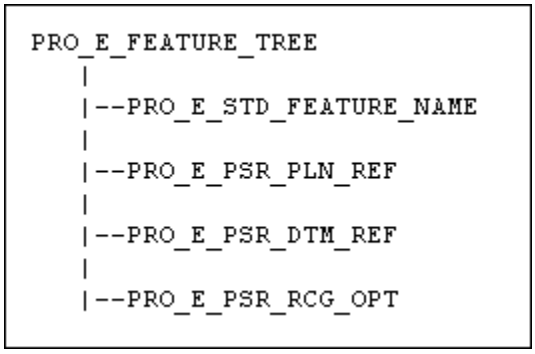
Note

The group header of the Planar Symmetry Recognition feature behaves as a standard feature. You can extract the header element tree in Creo Parametric TOOLKIT.

The Element Tree for Planar Symmetry Recognition Feature

The element tree for the Planar Symmetry Recognition feature is documented in the header file `ProSymmetryRecognition.h`, and is shown in the following figure:

Element Tree for Planar Symmetry Recognition Feature



The following table describes the elements in the element tree for the Planar Symmetry Recognition feature:

Element ID	Data Type	Description
`PRO_E_STD_FEATURE_NAME`	`PRO_VALUE_TYPE_WSTRING`	Specifies the name of the feature. The default value is `Mirror_Recognition`.
`PRO_E_PSR_PLN_REF`	`PRO_VALUE_TYPE_SELECTION`	Mandatory element. Specifies the collection of two seed surfaces or one seed surface and the symmetry mirror plane.

Element ID	Data Type	Description
PRO_E_PSR_DTM_REF	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the collection of datums and curve chains that will be included in the symmetry recognition.
PRO_E_PSR_RCG_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of symmetrical geometry to be recognized: **Identical** or **Similar**. This element takes the following values: • PRO_PSR_IDENTICAL • PRO_PSR_SIMILAR

Attach Feature

This section describes how to construct and access the element tree for Attach feature. It also shows how to create, redefine, and access the properties of this feature.

Introduction

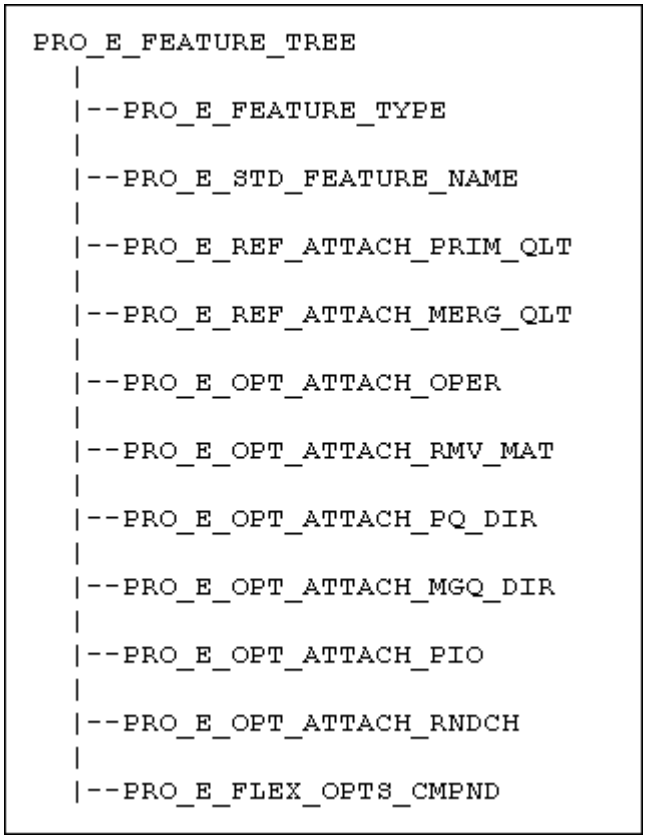
The Attach feature allows you to attach open quilts to solid or quilt geometry, if the open quilt does not intersect the solid or quilt geometry. You can select an open quilt and attach it to another quilt or solid geometry within the same model. You can also select two open quilts within the same model which do not intersect and attach them.

This feature is useful in case of UDF placement when the geometry of the UDF does not intersect the part.

The Element Tree for Attach Feature

The element tree for the Attach feature is documented in the header file `ProFlexAttach.h`, and is shown in the following figure:

Attach Feature Element Tree



The following table describes the elements in the element tree for the Attach feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of feature. The value of this feature must be PRO_FEAT_ATTACH.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Attach_1.
PRO_E_REF_ATTACH_PRIM_QLT	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the primary open quilt to be attached to another quilt. The primary quit will be extended or trimmed during the attachment.
PRO_E_REF_ATTACH_MERG_QLT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the merge quilt or solid geometry upto which the primary quilt can be extended or trimmed.
PRO_E_OPT_ATTACH_OPER	PRO_VALUE_TYPE_BOOLEAN	Mandatory element. Specifies whether to attach the primary quilt to the merge quilt or solid geometry.

Element ID	Data Type	Description
		This element takes the following values: • PRO_B_TRUE—Specifies that the primary quilt must not be attached. • PRO_B_FALSE—Specifies that the primary quilt must be attached.
PRO_E_OPT_ATTACH_RMV_MAT	PRO_VALUE_TYPE_BOOLEAN	This element is available when the element PRO_E_REF_ATTACH_MERG_QLT has no reference geometry specified. Specifies if material must be added or removed. This element takes the following values: • PRO_B_TRUE—Specifies that the material must be removed. • PRO_B_FALSE—Specifies that the material must be added.
PRO_E_OPT_ATTACH_PQ_DIR	PRO_VALUE_TYPE_BOOLEAN	This element is available when the element PRO_E_OPT_ATTACH_OPER has its value as PRO_B_FALSE. Specifies the side of the quilt that must be included in the merged quilt This element takes the following values: • PRO_B_TRUE—Specifies that the second side of the quilt must be merged. • PRO_B_FALSE—Specifies that the first side of the quilt must be merged.
PRO_E_OPT_ATTACH_MGQ_DIR	PRO_VALUE_TYPE_BOOLEAN	This element is available when the element PRO_E_OPT_ATTACH_OPER has its value as PRO_B_TRUE and the element PRO_E_REF_ATTACH_MERG_QLT has reference geometry specified. Specifies the direction in which the material should be added or removed in the primary quilt. This element takes the following values:

Element ID	Data Type	Description
		• PRO_B_TRUE—Specifies that the material must be added or removed from the second side. • PRO_B_FALSE—Specifies that the material must be added or removed from the first side.
PRO_E_OPT_ATTACH_PIO	PRO_VALUE_TYPE_BOOLEAN	Specifies if the quilt should be attached to the model geometry in the same way as it was attached previously by using the attachment information stored in the intent objects.
PRO_E_OPT_ATTACH_RNDCH	PRO_VALUE_TYPE_BOOLEAN	Specifies if the round or chamfer geometry of the quilt should be attached using the attachment information stored in the intent objects.
PRO_E_FLEX_OPTS_CMPND	Compound	Mandatory element that contains the flexible modeling geometry attachment options to attach the substituting surfaces. Specifies the integer and chain collection type elements. The elements related to reattachment of geometry in flexible modeling are defined in ProFlxmdlOpts.h. For more information, see the section Attachment Geometry Feature on page 1082.

Example 1: Creating a Flexible Model Feature

The sample code in UgFlexModelCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat shows how to create a Creo Flexible Modeling feature.

50

Element Trees: Bushing Load

This chapter describes how to construct and access the element tree for Bushing Load features in Creo Parametric TOOLKIT. It also shows how to redefine, create and access the properties of these features.

Introduction

The bushing loads in the model are represented under the **Bushing Load** node in the product mechanism tree. As the bushing load is a regular Creo Parametric feature, it also has an appropriate node in the main model tree. You can create, edit, and redefine the bushing loads. To create new bushing loads select a weld connection or a 6DOF connection. You must specify six stiffnesses and six damping coefficients for all the degrees of freedom of the reference connection. Each coefficient is associated with one feature parameter.

Note

- The default units of the spring stiffnesses and the damping coefficients on the three rotational axes are degree-based.
- If bushing load reference is a weld connection, any axis can be locked. In this case the spring stiffness on the locked axis should have a value of -1.0, regardless of the current units.

The Feature Element Tree for Bushing Loads

The element tree for the bushing load feature is documented in the header file `ProBushingLoadFeat.h`, and is shown in the following figure.

Element Tree for Bushing Loads

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_BUSHLD_REF
   |
   |--PRO_E_BUSHLD_T1_STF_COEFF
   |--PRO_E_BUSHLD_T1_DMP_COEFF
   |
   |--PRO_E_BUSHLD_T2_STF_COEFF
   |--PRO_E_BUSHLD_T2_DMP_COEFF
   |
   |--PRO_E_BUSHLD_T3_STF_COEFF
   |--PRO_E_BUSHLD_T3_DMP_COEFF
   |
   |--PRO_E_BUSHLD_R1_STF_COEFF
   |--PRO_E_BUSHLD_R1_DMP_COEFF
   |
   |--PRO_E_BUSHLD_R2_STF_COEFF
   |--PRO_E_BUSHLD_R2_DMP_COEFF
   |
   |--PRO_E_BUSHLD_R3_STF_COEFF
   |--PRO_E_BUSHLD_R3_DMP_COEFF
```

The following table describes the elements in the element tree for the bushing load feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature
PRO_E_BUSHLD_REF	PRO_VALUE_TYPE_ SELECTION	Specifies the weld or 6DOF reference connection
PRO_E_BUSHLD_T1_STF_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the spring stiffness on the 1st translation axis
PRO_E_BUSHLD_T1_DMP_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the damping coefficient on the 1st translation axis
PRO_E_BUSHLD_T2_STF_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the spring stiffness on the 2nd translation axis
PRO_E_BUSHLD_T2_DMP_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the damping coefficient on the 2nd translation axis
PRO_E_BUSHLD_T3_STF_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the spring stiffness on the 3rd translation axis
PRO_E_BUSHLD_T3_DMP_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the damping coefficient on the 3rd translation axis
PRO_E_BUSHLD_R1_STF_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the spring stiffness on the 1st rotational axis
PRO_E_BUSHLD_R1_DMP_ COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the damping coefficient on the 1st rotational axis

Element ID	Data Type	Description
PRO_E_BUSHLD_R2_STF_COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the spring stiffness on the 2nd rotational axis
PRO_E_BUSHLD_R2_DMP_COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the damping coefficient on the 2nd rotational axis
PRO_E_BUSHLD_R3_STF_COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the spring stiffness on the 3rd rotational axis
PRO_E_BUSHLD_R3_DMP_COEFF	PRO_VALUE_TYPE_DOUBLE	Specifies the damping coefficient on the 3rd rotational axis

51

Element Trees: Cosmetic Thread

This chapter describes how to construct and access the element tree for Cosmetic Thread features. It also shows how to create, redefine, and access the properties of these features.

Introduction

A cosmetic thread is a cosmetic feature that represents the diameter of a thread. It is displayed in purple. Unlike other cosmetic features, you cannot modify the line style of a cosmetic thread.

You can create cosmetic threads using cylinders, cones, splines, and non-normal planes as the references. The surface that you select determines whether a cosmetic thread is external or internal. If the surface is a shaft, the thread is external. If the surface is a hole, the thread is internal.

You can create a standard thread or a nonstandard thread. When you choose to create a standard thread, the standard thread series and diameter is used. You can create, edit, and redefine the cosmetic thread features.

The Element Tree for Cosmetic Thread

The element tree for the cosmetic thread feature is documented in the header file `ProThread.h`, and is shown in the following figure.

Element Tree for Cosmetic Thread

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_FEATURE_FORM
   |
   |-- PRO_E_STD_FEATURE_NAME
   |
   |-- PRO_E_THRD_TYPE_OPT_NEW
   |
   |-- PRO_E_THRD_SERIES_NEW
   |
   |-- PRO_E_THRD_SCREWSIZE_NEW
   |
   |-- PRO_E_THRD_SURF_NEW
   |
   |-- PRO_E_THRD_DIAM_NEW
   |
   |-- PRO_E_THRD_START_REF_NEW
   |
   |-- PRO_E_THRD_DEP_COMP_NEW
   |       |
   |       |-- PRO_E_THRD_DEP_OPT_NEW
   |       |
   |       |-- PRO_E_THRD_FLIP_OPT_NEW
   |       |
   |       |-- PRO_E_THRD_DEP_VAL_NEW
   |       |
   |       |-- PRO_E_THRD_END_REF
   |
   |-- PRO_E_THRD_NOTE_PARAMS_NEW
```

The following table describes the elements in the element tree for the cosmetic thread feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature
PRO_E_FEATURE_FORM	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of the feature form. Use the value PRO_EXTRUDE or PRO_REVOLVE from the enumerated type ProFeatFormType depending on the threaded surface. • Use PRO_EXTRUDE when the referenced threaded surface is cylindrical. • Use PRO_REVOLVE when the referenced threaded surface is conical.

Element ID	Data Type	Description
		Note If the enumerated value and referenced threaded surface are incorrectly selected, the feature creation or redefinition will fail.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifics the cosmetic thread feature name. The default value is COSMETIC_THREAD.
PRO_E_THRD_TYPE_OPT_NEW	PRO_VALUE_TYPE_BOOLEAN	Optional element. Specifies if a simple or a standard thread should be created. It has two values: FALSE for simple thread and TRUE for standard thread.
PRO_E_THRD_SERIES_NEW	PRO_VALUE_TYPE_INT	Specific the thread series for the standard threads. From the *.hol files information about different THREAD_SERIES is gathered and a list is generated. You can specify UNC, UNF, and ISO as the standard thread series. The current index to the list is stored in this element.
PRO_E_THRD_SCREWSIZE_NEW	PRO_VALUE_TYPE_INT	Optional element. Specifies the screw size for the standard thread. The screw_size list is extracted from the *.hol files corresponding to the thread series. The index to the screw_size list is stored in this element.
PRO_E_THRD_SURF_NEW	PRO_VALUE_TYPE_SELECTION	Specifies the surface to thread.
PRO_E_THRD_DIAM_NEW	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the diameter for a simple thread. If the reference threaded surface is conical, the thread diameter value can be between 0.0 and diam/4, where diam is the largest estimated diameter of the cone. For all other reference surfaces, the diameter can have values between 0.1 and MAX_DIM_VALUE, that is the maximum allowed diameter value.
PRO_E_THRD_START_REF_NEW	PRO_VALUE_TYPE_SELECTION	Specifies the starting location of the cosmetic thread.
PRO_E_THRD_DEP_COMP_NEW	Compound	Specifies the depth and flip elements. It contains the following elements:

Element ID	Data Type	Description
		• PRO_E_THRD_DEP_OPT _NEW • PRO_E_THRD_FLIP_OPT _NEW • PRO_E_THRD_DEP_VAL _NEW • PRO_E_THRD_END_REF
PRO_E_THRD_DEP_OPT_NEW	PRO_VALUE_TYPE_BOOLEAN	Optional element. Specifies the depth option: FALSE for blind depth option and TRUE for the depth up to the selected entity.
PRO_E_THRD_FLIP_OPT_NEW	PRO_VALUE_TYPE_INT	Optional element. Specifies the flip direction of the thread with respect to the reference surface. The flip direction is specified by the enumerated type ProThreadFlip that takes the following values: • PRO_COSTHREAD_THD _FLIP_OPT_FLIP • • PRO_COSTHREAD_THD _FLIP_OPT_NORM
PRO_E_THRD_DEP_VAL_NEW	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the depth value when the element PRO_E_THRD_DEP_OPT_NEW is FALSE (Blind option). You can specify a minimum depth of 0.1. When the element PRO_E_ THRD_DEP_OPT_NEW is TRUE, the node is invisible.
PRO_E_THRD_END_REF	PRO_VALUE_TYPE_ SELECTION	Mandatory element when PRO_ E_THRD_DEP_OPT_NEW is TRUE. Specifies the reference surface up to which the thread depth should be created. This node is available only when the element PRO_E_THRD_DEP_OPT_NEW is TRUE. When the element PRO_ E_THRD_DEP_OPT_NEW is FALSE, the node is invisible.
PRO_E_THRD_NOTE_ PARAMS_NEW	PRO_VALUE_TYPE_POINTER	Reserved for future use. Optional element. Displays the thread parameters in a data structure.

52

Element Trees: ECAD Area Feature

This chapter describes how to access ECAD Area feature through Creo Parametric TOOLKIT.

Introduction to ECAD Area Feature

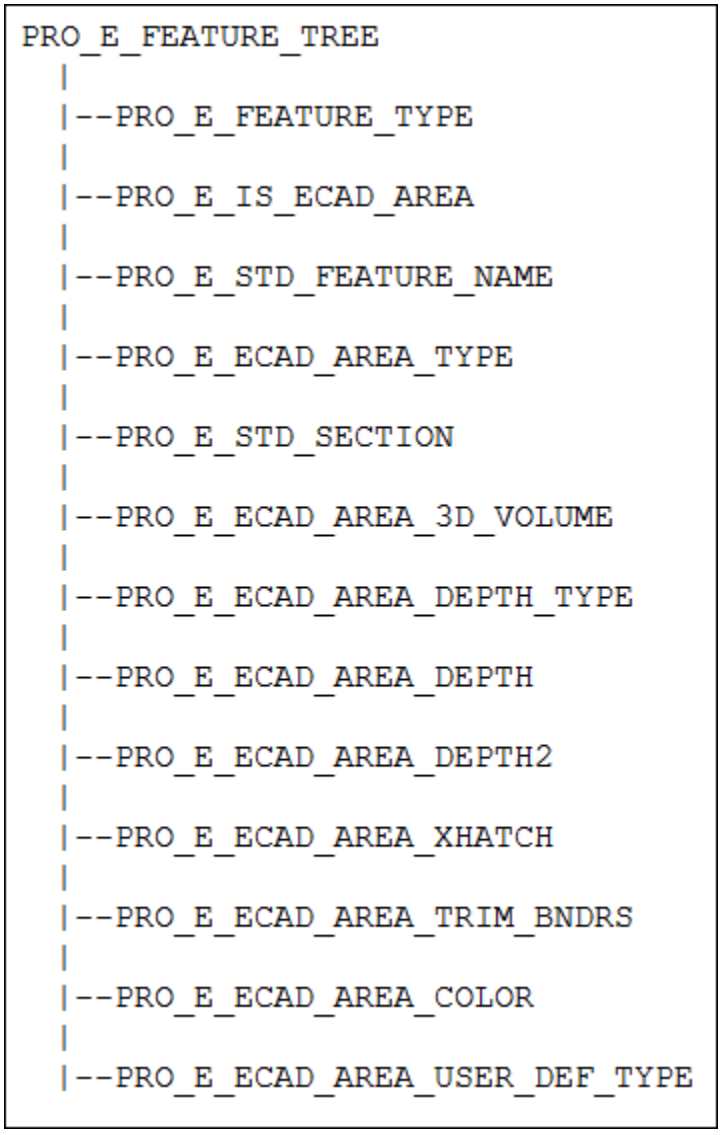
An ECAD Area specifies where you can place electrical components or cannot place them to avoid interference with other electric components or electrical routing. You can create an ECAD area as a sketched cosmetic feature of an electrical board part. However, since this area is sketched, you cannot dimension or regenerate it.

The ECAD area can have a closed 3D volume represented by a quilt. Use these quilts to perform a clearance and interference check on the neighboring electric components. You can access the geometry of the ECAD area feature by using standard Creo Parametric TOOLKIT functions such as `ProFeatureGeomitemVisit()` and `ProSolidQuiltVisit()`.

Feature Element Tree for the ECAD Area

The element tree for the ECAD Area feature is documented in the header file `ProEcadArea.h`. The following figure demonstrates the element tree structure:

Feature Element Tree for an ECAD Area



The information about the elements in this tree is as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type; should always have the value `PRO_FEAT_COSMETIC`.
- `PRO_E_IS_ECAD_AREA`—Specifies whether the created cosmetic feature is an ECAD area. This element must have the value `PRO_B_TRUE` to distinguish this feature from other cosmetic features.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the ECAD area feature.

- PRO_E_ECAD_AREA_TYPE—Specifies the type of ECAD area in the form of the enumerated type ProEcadAreaType. Specify the value of this element as one of the following:
  - PRO_ECAD_AREA_TYPE_PLACE_KEEPIN—Specifies a keepin region. You can specify the depth of the area above and below the electric board while creating this region. The default depth value is zero.
  - PRO_ECAD_AREA_TYPE_PLACE_KEEPOUT—Specifies a keepout region. You can specify the depth of the area above and below the electric board while creating this region. The default depth value is zero.
  - PRO_ECAD_AREA_TYPE_PLACE_REGION—Specifies a group area section.
  - PRO_ECAD_AREA_TYPE_ROUTE_KEEPIN—Specifies a routing keepin region where routing actions are permitted.
  - PRO_ECAD_AREA_TYPE_ROUTE_KEEPOUT—Specifies a routing keepout region where routing actions are not permitted.
  - PRO_ECAD_AREA_TYPE_VIA_KEEPOUT—Specifies an area where you cannot create vias.
  - PRO_ECAD_AREA_TYPE_FLEX_REGION—Specifies a region created using flexible modeling features.
  - PRO_ECAD_AREA_TYPE_USER_DEFINED—Specifies an user-defined area.
- PRO_E_STD_SECTION—Specifies the sketched region for the ECAD area. For more information on how to populate the section elements of the sketched region, refer to the Element Trees: Sketched Features on page 1004 chapter.
- PRO_E_ECAD_AREA_3D_VOLUME—A ProBoolean element that specifies whether the ECAD area appears with a 3D quilt. Only PRO_ECAD_AREA_TYPE_PLACE_KEEPIN and PRO_ECAD_AREA_TYPE_PLACE_KEEPOUT types of ECAD areas accept a 3D volume on one or two sides.
- PRO_E_ECAD_AREA_DEPTH_TYPE—Specifies the depth type in the form the enumerated type ProEcadAreaDepthType. The depth options are applicable only if the element PRO_E_ECAD_AREA_3D_VOLUME is set to PRO_B_TRUE. These options are as follows:
  - PRO_ECAD_AREA_DEPTH_ONE_SIDE—Creates the ECAD area with a 3D volume on one side of the electrical board.
  - PRO_ECAD_AREA_DEPTH_TWO_SIDES_SYM—Places the created ECAD area with a 3D volume at the top and bottom of the electrical board symmetrically.

- PRO_ECAD_AREA_DEPTH_TWO_SIDES_NOT_SYM—Places the created ECAD area with a 3D volume at the top and bottom of the electrical board asymmetrically.

- PRO_E_ECAD_AREA_DEPTH—Specifies the depth for the ECAD area.
- PRO_E_ECAD_AREA_DEPTH2—Specifies the other depth value, which is applicable only if the element PRO_E_ECAD_AREA_DEPTH_TYPE is set to PRO_ECAD_AREA_DEPTH_TWO_SIDES_NOT_SYM.
- PRO_E_ECAD_AREA_XHATCH—Specifies whether the ECAD area is created as a meshed region.
- PRO_E_ECAD_AREA_TRIM_BNDRS—Specifies whether to trim the boundary of an ECAD area, where the ECAD area exceeds the boundary of the electric board.
- PRO_E_ECAD_AREA_COLOR—Specifies the color of the ECAD area. The default color of the ECAD area is set in the **Creo Parametric Options** dialog box.

  Use the function ProElementEcadAreaProColorSet() to set the color of the ECAD area. Specify a defined ProColor structure as input argument. The data from ProColor structure is copied to the element PRO_E_ECAD_AREA_COLOR. When you redefine the element tree, the data copied from ProColor structure is used to set the color of the ECAD area.

  The function ProElementEcadAreaProColorGet() returns the color of the specified ECAD area.
- PRO_E_ECAD_AREA_USER_DEF_TYPE—Specifies the name of the user-defined area. Similar to the other types of ECAD areas, the color of the user-defined ECAD area is also set using the element PRO_E_ECAD_AREA_COLOR.

  You can also set the name and color of user-defined area in a comma separated value (.csv) file. The .csv file contains 2 columns, the names of user-defined areas and the names of the colors associated with each area. The names of the colors set in the .csv file must be defined in the *.dmt file.

  Set the path to the .csv file using the configuration option ecad_usrdef_area_type_file_path.

  If you specify the name of a user-defined area from the .csv file, then the ECAD area is created with the color specified for that area in the .csv file. For an user-defined area that was created from the .csv file, if you use the element PRO_E_ECAD_AREA_COLOR to set the area color, then the color set by the element overrides the color set by the *.csv file.

  Refer to the Creo Parametric ECAD Help for more information on how to create the .csv file and set colors in *.dmt files.

53

Assembly: Basic Assembly Access

This chapter describes the Creo Parametric TOOLKIT functions that access the contents of a Creo Parametric assembly. Before you read this chapter, you should be familiar with the following documentation:

- User Interface: Selection on page 503
- Core: Coordinate Systems and Transformations on page 222
- Core: 3D Geometry on page 170

Structure of Assemblies and Assembly Objects

The object `ProAssembly` is an instance of `ProSolid` and shares the same declaration. The `ProAssembly` object can therefore be used as input to any of the `ProSolid` and `ProMdl` functions applicable to assemblies. In particular, because you can use the function `ProSolidFeatVisit()` to traverse features, you can extract the assembly datum features and their geometry in the same way as for parts (described in detail in the chapter on Core: 3D Geometry on page 170).

However, assemblies do not contain active geometry items other than those in datums—that is, no "solid" geometry as described in the Core: 3D Geometry on page 170 and Element Trees: Principles of Feature Creation on page 764 chapters. Therefore, the function `ProSolidBodySurfaceVisit()` will not find any surfaces, and solid assembly features such as holes and slots will not contain active surfaces or edges.

The solid geometry of an assembly is contained entirely in its components. Each component is a feature of type `PRO_FEAT_COMPONENT`, which is a reference to a part or another assembly, plus a set of parametric constraints for determining its geometric location within the parent assembly.

Assembly features that are solid, such as holes and slots, and therefore affect the solid geometry of parts in the assembly hierarchy, do not themselves contain the geometry items that describe those modifications. These items are always contained in the parts whose geometry is modified, within local features created for that purpose of type `PRO_FEAT_ASSEM_CUT`.

The most important Creo Parametric TOOLKIT functions for assemblies are those that operate on the components of an assembly. The object `ProAsmcomp`, which is an instance of `ProFeature` and shares its DHandle declaration, is defined for that purpose. Each assembly component is treated as a variety of feature, and the integer identifier of the component is also the feature identifier.

An assembly component can be another assembly. In general, therefore, an assembly can contain a hierarchy of assemblies and parts at many levels, in which some assemblies and parts may appear more than once. To identify the role of any database item in the context of the root assembly, it is not enough to have the integer identifier of the item and the handle to its owning part or assembly, as would be provided by its `ProFeature` or `ProGeomitem` description. It is also necessary to give the full path of the assembly-component references down from the root assembly to the part or assembly that owns the database item. The object `ProAsmcomppath`, which is used as the input to Creo Parametric TOOLKIT assembly functions, accomplishes this purpose.

The declaration of `ProAsmcomppath` is as follows:

```
typedef struct pro_comp_path
{
```

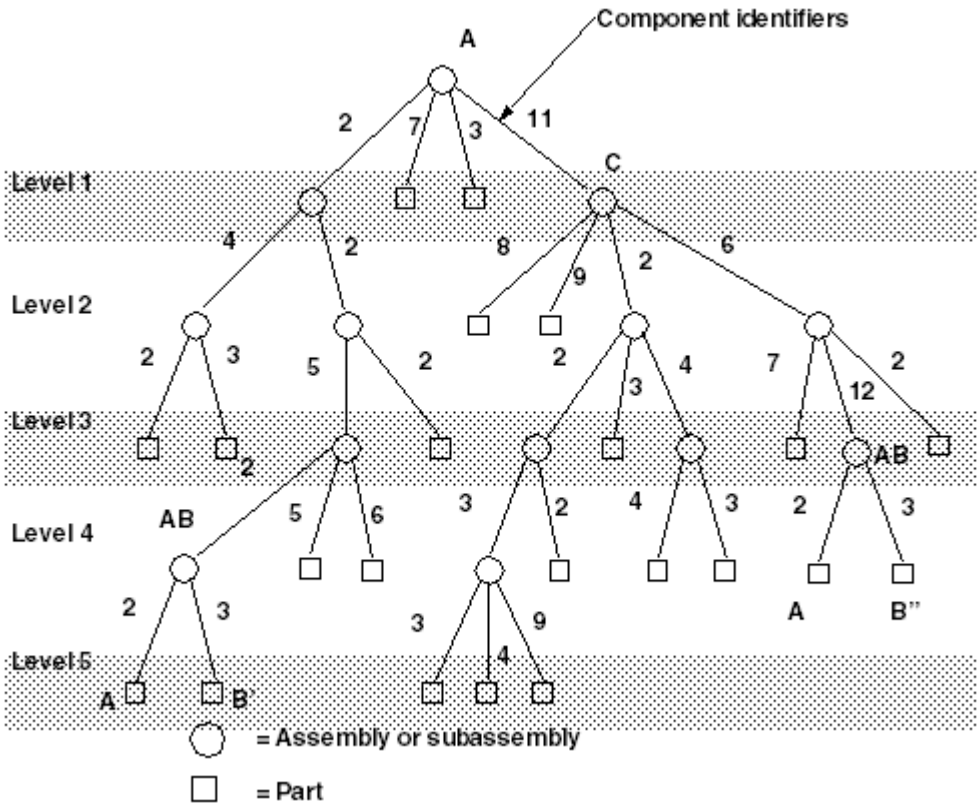
```
    ProSolid    owner;
    ProIdTable  comp_id_table;
    int         table_num;
  } ProAsmcomppath;
```

The data structure fields are as follows:

- `owner`—Identifies the root assembly
- `comp_id_table` (the component identifier table)—An integer array that contains the identifiers of the components that form the path from the root assembly down to the component part or assembly being referred to
- `table_num`—Specifies the number of component identifiers in the `comp_id_table` array

The following figure Sample Assembly Hierarchy on page 1132 shows an assembly hierarchy with two examples of the contents of a `ProAsmcomppath` object.

Sample Assembly Hierarchy



In the assembly shown in Figure 12-1, Sample Assembly Hierarchy on page 1132 subassembly C is component identifier 11 within assembly A, Part B is component identifier 3 within assembly AB, and s on. The subassembly AB occurs twice. To refer to the two occurrnces of part B, use the following:

```
Component B'            Component B"
table_num = 5           table_num = 4
comp_id_tab[0] = 2      comp_id_tab[0] = 11
comp_id_tab[1] = 2      comp_id_tab[1] = 6
comp_id_tab[2] = 5      comp_id_tab[2] = 12
comp_id_tab[3] = 2      comp_id_tab[3] = 3
comp_id_tab[4] = 3
```

A `ProAsmcomppath` structure in which `table_num` is set to 1 contains the same information as a `ProAsmcomp` object.

The object `ProAsmcomppath` is one of the main ingredients in the `ProSelection` object, as described in The Selection Object on page 504.

Visiting Assembly Components

Functions Introduced:

- **ProSolidFeatVisit()**
- **ProFeatureTypeGet()**

Each component of an assembly is also a feature of that assembly. Therefore, to visit the components, visit the features using `ProSolidFeatVisit()` and find those features whose type is `PRO_FEAT_COMPONENT` using the function `ProFeatureTypeGet()`. You can convert the `ProFeature` object for each component to the `ProAsmcomp` object by casting.

Properties Related to Component Purpose

Functions Introduced:

- **ProAsmcomppathInit()**
- **ProAsmcompMdlMdlnameGet()**
- **ProAsmcompMdlGet()**
- **ProAsmcomppathMdlGet()**
- **ProAsmcompTypeGet()**
- **ProAsmcompMdldataGet()**

To create a `ProAsmcomppath` object for the component, use the function `ProAsmcomppathInit()` and set the component identifier table to contain only a single component identifier.

The function `ProAsmcompMdlMdlnameGet()` retrieves the model name and type for the component. If an assembly component is missing on retrieval, the function `ProAsmcompMdlMdlnameGet()` still provides information about the component while the function `ProAsmcompMdlGet()` fails to retrieve a valid model handle.

The function `ProAsmcompMdlGet()` provides the `ProMdl` handle to the part or assembly being referenced by the component. To traverse the components at all levels in the assembly hierarchy, make a recursive function to perform these steps:

1. Call `ProAsmcompMdlGet()` for each component of the root assembly to find the model for the component.
2. Call `ProMdlTypeGet()` to find out if the model is a part or an assembly.
3. If the model is an assembly, traverse each component by calling `ProSolidFeatVisit()` again.

The function `ProAsmcomppathMdlGet()` retrieves a model specified by `ProAsmcomppath` and is useful when analyzing a `ProSelection` object that refers to an assembly.

The function `ProAsmcompTypeGet()` yields the type of the assembly component. Examples of the possible types are as follows:

- `PRO_ASM_COMP_TYPE_WORKPIECE`—Workpiece
- `PRO_ASM_COMP_TYPE_REF_MODEL`—Reference model
- `PRO_ASM_COMP_TYPE_FIXTURE`—Fixture
- `PRO_ASM_COMP_TYPE_MOLD_BASE`—Mold base
- `PRO_ASM_COMP_TYPE_MOLD_COMP`—Mold component
- `PRO_ASM_COMP_TYPE_MOLD_ASSEM`—Mold assembly
- `PRO_ASM_COMP_TYPE_GEN_ASSEM`—General assembly
- `PRO_ASM_COMP_TYPE_CAST_ASSEM`—Cast assembly
- `PRO_ASM_COMP_TYPE_DIE_BLOCK`—Die block
- `PRO_ASM_COMP_TYPE_DIE_COMP`—Die component
- `PRO_ASM_COMP_TYPE_SAND_CORE`—Sand core
- `PRO_ASM_COMP_TYPE_CAST_RESULT`—Cast result
- `PRO_ASM_COMP_TYPE_FROM_MOTION`—Component for use by Creo Simulate.
- `PRO_ASM_COMP_TYPE_NO_DEF_ASSUM`—Component for which Creo Parametric cannot apply default assumptions.

The function `ProAsmcompMdldataGet()` takes the handle to the assembly component as its input argument and retrieves the following information:

- *r_mdl_type*—Specifies the type of the model using the enumerated data type `ProMdlType`.
- *r_mdl_filetype*—Specifies the file type of the component using the enumerated data type `ProMdlfileType`.
- *r_mdl_name*—Specifies the name of the component. You must free this argument using the function `ProWstringFree()`.

Component Placement

- **ProAsmcompIsBulkitem()**
- **ProAsmcompIsPackaged()**

The function `ProAsmcompIsBulkitem()` reports whether an assembly component is a bulk item. A bulk item is a non-geometric assembly feature that should appear in an assembly bill of materials.

Use the function `ProAsmcompIsPackaged()` to determine whether the specified component is packaged.

Simplified Representations

- **ProAsmcompIsUnderconstrained()**
- **ProAsmcompIsFrozen()**
- **ProAsmcompIsUnplaced()**
- **ProAsmcompIsPlaced()**
- **ProAsmcompIsSubstitute()**
- **ProAsmcompVisibilityGet()**

The function `ProAsmcompIsUnderconstrained()` determines whether the specified component is underconstrained, that is, it has one or more constraints but they are not sufficient to fully constraint the component location.

The function `ProAsmcompIsFrozen()` determines whether the specified component is frozen. The frozen component behaves similar to the packaged component and does not follow the constraints that you specify.

From Creo Parametric 3.0 onward, the frozen status in components is set only during the regeneration of the model. You cannot use the freeze commands in the Creo Parametric user interface, to set the frozen status on a component. By default, behavior of the configuration option `freeze_failed_assy_comp` is ignored. For the models created in releases prior to Creo Parametric 3.0, the frozen status of components is retained during model retrieval.

The configuration option `allow_freeze_failed_assy_comp` allows you to restore the behavior of the configuration option `freeze_failed_assy_comp` and freeze commands in the Creo Parametric user interface. The valid values are:

- `yes`—Specifies that the behavior of the configuration option `freeze_failed_assy_comp` is available. The freeze commands in the Creo Parametric user interface are also available. The valid values for the configuration option `freeze_failed_assy_comp` are:
  - `yes`—Automatically freezes any component that fails retrieval into the assembly at its last known location.
  - `no`—Requires you to perform specific actions to fix the assembly or freeze the component that fails retrieval.
- `no`—This is the default value. Specifies that the behavior of the configuration option `freeze_failed_assy_comp` is ignored.

The functions `ProAsmcompIsUnplaced()` and `ProAsmcompIsPlaced()` determine whether the specified component is unplaced or placed respectively. Unplaced components belong to an assembly without being assembled or packaged.

The function `ProAsmcompIsSubstitute()` determines whether the specified component is substituted. When you substitute a component in a simplified representation, you temporarily exclude the substituted component and superimpose the substituting component in its place.

The function `ProAsmcompVisibilityGet()` to skip components of the master representation that are not shown in the representation when you traverse the assembly components of a simplified representation.

Modifying Component Properties

- **ProAsmcompTypeSet()**
- **ProAsmcompFillFromMdl()**
- **ProAsmcompMakeUniqueSubasm()**
- **ProAsmcompRmvUniqueSubasm()**
- **ProAsmcompSetPlaced()**

The function `ProAsmcompTypeSet()` enables you to set the type of a component.

The function `ProAsmcompFillFromMdl()` copies the template model into the model of the component.

Note

The function returns the error `PRO_TK_UNSUPPORTED` when the model to which the template model is being copied is an unsupported model. For example, it is an embedded model.

Use the function `ProAsmcompMakeUniqueSubasm()` to create a unique instance of a sub-assembly by specifying the path of the sub-assembly. The function `ProAsmcompRmvUniqueSubasm()` removes the instance of the sub-assembly.

The function `ProAsmcompSetPlaced()` forces Creo Parametric to consider a particular component to be placed or unplaced.

Example 1: Listing the Members of an Assembly

The sample code in the file `UgAsmCompVisit.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_asmt`, recursively lists the components of an assembly and writes the name of the solid for each component.

Locations of Assembly Components

Functions Introduced:

- **ProAsmcomppathTrfGet()**
- **ProAsmcomppathTrfSet()**
- **ProAssemblyDynPosGet()**
- **ProAssemblyDynPosSet()**
- **ProAsmpathProarrayFree()**

The function `ProAsmcomppathTrfGet()` provides the transformation matrix that describes the coordinate transformation between the coordinate system of an assembly component and that of the root assembly. As its name implies, its input is a `ProAsmcomppath` object, so it can be applied to a component at any level within an assembly hierarchy. It has an option to provide the transformation from bottom to top, or from top to bottom. (To apply the transformation, use the function `ProPntTrfEval()` or `ProVectorTrfEval()`, described in the section Coordinate Systems on page 223.)

In effect, this function describes the current position and orientation of the assembly component in the root assembly.

Use the function `ProAsmpathProarrayFree()` to free the memory allocated to the `ProArray` of type `ProAsmpath`.

Example 2: Finding the Position of a Component

The sample code in the file `UgAsmcompTransfGet.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_asm` shows a function that finds the matrix that describes the position of an assembly component in its parent assembly.

Assembling Components

To assemble components into an assembly, use the methods of feature creation. These methods are described in detail in the chapter Assembly: Assembling Components on page 1159.

Redefining and Rerouting Components

The functions used to redefine and reroute components are described in the chapter Assembly: Assembling Components on page 1159.

Deleting Components

Function Introduced:

- **ProFeatureDelete()**

The function `ProFeatureDelete()` deletes components. It has the same options as described in the sectionchapter Core: Features on page 131.

Flexible Components

A flexible component allows variance of items such as features, dimensions, annotations, and parameters of a model in the context of an assembly.

The object `ProAsmitem` describes the contents of a variant item in an assembly component. The declaration for this object is as follows:

```
typedef struct pro_asm_item
{
   ProModelitem item;
   ProName      name; /* used for PRO_PARAMETER in
                         this case item->type == PRO_PART or PRO_ASSEMBLY */
   ProAsmcomppath path;
}  ProAsmitem;
```

Refer to the section Exploded State Objects on page 1142 for the declaration of the ProModelitem object. The field name in the ProAsmitem object is used only in case of PRO_PARAMETER; wherein the field type in the ProModelitem object is either PRO_PART or PRO_ASSEMBLY.

In case of parameter initialization, the field name specifies the parameter name and the fields type and id in the ProModelitem object initiate the model on which the parameter is defined. For non-parameter items such as features, dimensions, and annotations, the fields owner, type and id in the ProModelitem object have the same values as the values of the input arguments for the function ProModelitemInit().

The field path specifies the path from the top-level component model. This field is empty if the variant items are defined on the top-level component model.

Functions Introduced:

- **ProAsmcompAsmitemInit()**
- **ProAsmcompFlexibleSet()**
- **ProAsmcompFlexibleUnset()**
- **ProAsmcompIsFlexible()**
- **ProAsmcompFlexiblemodelAdd()**
- **ProAsmcompVarieditemsToModelAdd()**
- **ProAsmcompFlexibleWithPredefineditemsSet()**

Use the function ProAsmcompAsmitemInit() to initialize the ProAsmitem object that describes the contents of a variant item in an assembly component.

The function ProAsmcompFlexibleSet() converts a specified assembly component to a flexible component based on an array of specified variant items.

Use the function ProAsmcompFlexibleUnset() to convert a flexible assembly component to a regular component.

Use the function ProAsmcompIsFlexible() to identify if the specified assembly component is a flexible component. The function returns PRO_B_TRUE if the component is a flexible component, otherwise it returns PRO_B_FALSE.

The function ProAsmcompFlexiblemodelAdd() creates a flexible model from the specified model of the flexible component.

Note

The model is temporarily converted into a flexible model in order to allow you to define variant items on it using the existing Creo Parametric TOOLKIT functions for variant items. If no variant items are added, the temporary flexible model becomes a regular one upon regeneration. You can convert a model into a temporary flexible model only via Creo Parametric TOOLKIT. For information on the functions that can be used to access and modify the variant items in a flexible assembly component, refer to the section Inheritance Feature and Flexible Component Variant Items on page 1215.

The function `ProAsmcompVarieditemsToModelAdd()` adds an array of specified variant items to the predefined flexibility definition of the specified component model. All varied items are replaced by the provided ones.

The function `ProAsmcompFlexibleWithPredefineditemsSet()` converts a specified assembly component to a flexible component using the predefined flexibility definition of the variant items in the component model.

Embedded Components and Inseparable Assemblies

An embedded component is a copy of the component model that becomes a part of its parent assembly. The parent assembly can be the top-level assembly or a subassembly.

In an inseparable assembly some components are embedded in the assembly. It is a single file that contains the embedded components in the assembly. The assembly structure is visible in the **Model Tree**.

The inseparable assembly is created in the open session of Creo Parametric and must be saved to be used again. When you embed a component in an assembly, you copy the model to the assembly file. The original model is not affected and continues to exist in your database. The new embedded copy is not dependent on the original model. If you no longer need the original model, you can erase it from the session and remove it from the database. When you embed a component, the name of the embedded copy is appended to include the name of the owner assembly.

Functions Introduced:

- **ProAsmcompEmbed()**
- **ProAsmcompExtract()**
- **ProAsmcompEmbeddedOwnerMdlGet()**

Use the function `ProAsmcompEmbed()` to embed selected components in its owner assembly. The input arguments follow:

- *comp_sel*—Selection of components specified using the array of `ProSelection` object.
- *embed_recursively*—`ProBoolean` that is used only when the input argument *comp_sel* is a subassembly selection.
    - If the value of *embed_recursively* is set to `PRO_B_FALSE` and *comp_sel* is a subassembly selection, only the subassembly is embedded.
    - If the value of *embed_recursively* is set to `PRO_B_TRUE` and *comp_sel* is a subassembly selection, the subassembly and all the possible components are embedded.

The function `ProAsmcompExtract()` extracts the embedded component from the owner assembly. The input argument follow:

- *comp_sel*—Selection of components specified using the array of `ProSelection` object.
- *newMdlName*—Name of the new model.

The extracted component becomes a standalone model and replaces the embedded component in the assembly. This name of the new model is specified using the input argument *newMdlName*.

The function returns the error `PRO_TK_NO_CHANGE` if the selected component is not embedded in the owner assembly.

The function `ProAsmcompEmbeddedOwnerMdlGet()` returns the handle of the nonembedded owner model for the specified embedded model.

Refer to the Creo Parametric online help, for more information about Inseparable Assemblies and Embedded Components.

Exploded Assemblies

An exploded view of an assembly shows each component of the model separated from the other components. An exploded view affects the appearance of the assembly only. The design intent and the true distance between the assembly components do not change.

Functions Introduced:

- **ProAssemblyExplode()**
- **ProAssemblyUnexplode()**
- **ProAssemblyIsExploded()**

The functions `ProAssemblyExplode()` and `ProAssemblyUnexplode()` enable you to explode and unexplode an assembly. The function `ProAssemblyIsExploded()` identifies whether the specified assembly is exploded. Use this function in the assembly mode only.

The exploded status of an assembly depends on the mode. If an assembly is opened in the drawing mode, the state of the assembly in the drawing view is displayed. The drawing view does not represent the actual exploded state of the assembly. Use the function `ProDrawingViewExplodedGet()` to get the exploded state of an assembly for a specified drawing view.

Note

These functions explode the assembly using the default exploded state of the assembly. Creo Parametric defines the positions of an assembly component in the default exploded state.

Exploded State Objects

The structure `ProExpldstate` describes the contents of an exploded state object. This object uses the same declaration as the `ProModelitem`, `ProGeomitem`, and `ProFeature` objects, which is as follows:

```
typedef struct pro_model_item
{
  ProType  type;
  int      id;
  ProMdl   owner;
} ProExpldstate;
```

Visiting Exploded States

Function Introduced:

- **ProSolidExpldstateVisit()**

The function `ProSolidExpldstateVisit()` enables you to visit all the exploded states in the specified solid, except for the default exploded state. For a detailed explanation of visiting functions, see the section Visit Functions on page 62 in the Fundamentals on page 22 chapter.

Accessing Exploded States

Functions Introduced:

- **ProExpldstateInit()**
- **ProExpldstateActivate()**
- **ProExpldstateSelect()**
- **ProExpldstateActiveGet()**
- **ProExpldstateNameGet()**
- **ProExpldstateNameSet()**
- **ProExpldstateExplodedcomponentsGet()**
- **ProExpldStateExplodeLinesGet()**
- **ProExpldstateMovesGet()**
- **ProExpldstateMovesSet()**
- **ProExpldAnimDataTranslatemoveInit()**
- **ProExpldAnimDataRotatemoveInit()**
- **ProExpldanimmovedataProarrayFree()**

The function `ProExpldstateInit()` returns the handle to a specified exploded state representation of a solid. It takes the following input arguments:

- *expld_name*—Specifies the name of the exploded state. If you specify this value, then the function ignores the next argument *expld_id*.
- *expld_id*—Specifies the identifier of the exploded state. This argument is applicable only if the argument *expld_name* is `NULL`.
- *p_solid*—Specifies the Creo Parametric solid that contains the exploded state. This argument cannot be `NULL`.

The function `ProExpldstateActivate()` activates a specified exploded state representation of a solid.

The function `ProExpldstateSelect()` enables you to select a specific exploded state from the list of defined exploded states.

The function `ProExpldstateActiveGet()` retrieves the current active exploded state for the specified solid.

The functions `ProExpldstateNameGet()` and `ProExpldstateNameSet()` return and set, respectively, the name of the exploded state.

The function `ProExpldstateExplodedcomponentsGet()` returns an array of assembly component paths that are included in the exploded state.

The function `ProExpldStateExplodeLinesGet()` returns an array of explode lines for the specified exploded state.

The functions `ProExpldstateMovesGet()` and `ProExpldstateMovesSet()` retrieve and assign, respectively, the array of moves of an exploded state.

In order to define an exploded position of an assembly component (or a set of assembly components), you need to perform a sequence of moves. For example, you can move the assembly component over the X-axis, rotate over a selected edge, and then move over the Y-axis. In this case, the final position of the assembly component (or a set of assembly components) is attained by three moves.

The `ProExpldAnimMoveData` object describes the moves of an exploded state. The fields in this object are as follows:

- `comp_set`—Specifies an array of paths of the assembly components, in the form of the `ProAsmcomppath` objects.
- `move`—Specifies the move of the exploded state. It is given by the `ProExpldAnimMove` object. This object contains the following fields:
  - `move_type`—Specifies the move type in terms of the enumerated type `ProExpldAnimMoveType`. The move can be one of the following types:
    - `PRO_EXPLDANIM_MOVE_TRANSLATE`
    - `PRO_EXPLDANIM_MOVE_ROTATE`
  - `direction`—Depending upon the selected move type, this field specifies the translation direction or the rotational axis.
  - `value`—Depending upon the selected move type, this field specifies the translational distance or the rotation angle.

The function `ProExpldAnimDataTranslatemoveInit()` creates a translational move based on the specified direction of the translation and the specified array of the assembly components to which this move is applied.

The function `ProExpldAnimDataRotatemoveInit()` creates a rotational move based on the specified rotational axis, rotation angle, and the specified array of the assembly components to which this move is applied.

The function `ProExpldanimmovedataProarrayFree()` clears the array of assigned `ProExpldAnimMoveData` objects.

Manipulating Exploded States

Functions Introduced:

- **ProExpldstateCreate()**
- **ProExpldstateDelete()**

The function `ProExpldstateCreate()` creates a new exploded state based on the values of the following input arguments:

- *p_solid*—Specifies the assembly in which the exploded state is created. This argument cannot be `NULL`.
- *name*—Specifies the name of the exploded state. This argument also cannot be `NULL`.
- *p_move_arr*—Specifies an array of `ProExpldAnimMoveData` objects.

Use the function `ProExpldstateDelete()` to delete a specified exploded state.

Merge and Cutout

The Merge and Cutout function has been deprecated. Use the Merge feature element tree to create merge or cutout feature. See the section Merge Feature on page 861 in Element Trees: Edit Menu Features on page 853 for more details.

Automatic Interchange

Functions Introduced:

- **ProAssemblyAutointerchange()**

In Creo Parametric, it is possible to interchange an assembly component with another model that contains equivalent assembly constraints. The Creo Parametric TOOLKIT function that performs this action is `ProAssemblyAutointerchange()`. Depending on the type of component interchange, the assembly constraints may need to be respecified for the replacement model.

Instances in a family table share the same assembly constraints. Consequently, you can automatically replace an assembly component with another instance in the component's family table without respecifying any assembly constraints. Simply retrieve the handle for the replacement instance and pass this handle to `ProAssemblyAutointerchange()`.

If the assembly component and replacement model are not instances in the same family table, you can define the necessary relationships between them interactively and save them in an Interchange Assembly. (See the *Assembly Modeling User's Guide* for details.) To perform an interchange using models in an interchange assembly, first retrieve the interchange assembly (using the function `ProMdlnameRetrieve()`), and then pass the handle of the replacement model to function `ProAssemblyAutointerchange()`. Note that the interchange assembly must be in memory before the call to `ProAssemblyAutointerchange()`.

An interchange assembly is not the same as an interchange domain. Interchange domains (`.int` files) contain interchange information, but they can no longer be created using Creo Parametric. However, it is possible to use `ProAssemblyAutointerchange()` to interchange models using an existing interchange domain.

54

Assembly: Top-down Design

This chapter describes the Creo Parametric TOOLKIT Design Manager functions. For more information on Design Intent, Top-Down Design, and other Design Manager issues, refer to the Assembly portion of the Creo Parametric help data, or the Top-Down Design Task Guide.

Overview

Creo Parametric supports a design concept called Top-Down Design. Top-Down Design is a method of designing a product by specifying top-level design criteria and passing those criteria down from the top level of the product's structure to all affected subsystems. The Creo Parametric TOOLKIT Design Manager functions support this design concept. The next sections contain a brief summary of the six steps of Top-Down Design.

Defining Design Intent

Before building parts and assemblies, it is important that you define the intent of your design. Doing this means defining:

- Purpose or function of the product
- Major systems and subsystems required
- Incorporation of subsystems into the overall product
- Dependence (if any) on any existing design or product

Design criteria and parameters you specify in this process can be shared globally among all components of the assembly, and can be used to drive design parts, assemblies, and skeleton models.

Defining Preliminary Product Structure

The preliminary product structure consists of a list of components and their hierarchy within the assembly design. This structure allows creation of subassemblies and parts without requiring creation of geometry and without having to assemble parts. You can add existing subassemblies and parts to this structure. You can also define non-geometric information for the entire design and capture design parameters including description, part number, and part type. Creo Parametric TOOLKIT Design Manager manages the assembly structure with assembly component functions.

Introducing Skeleton Models

Skeleton models are a 3-dimensional layout of the assembly. These models are holders or distributors of critical design information, and can represent space requirements, important mounting locations, and motion.

Skeleton models can contain the master definition of the design information, which can be propagated to specific design models. You can also use skeleton models to communicate design information between components. Creo Parametric TOOLKIT Design Manager uses skeleton model functions to manipulate these models.

Communicating Design Intent Throughout the Assembly Structure

Designers can distribute top-level design information to dependent skeleton models in the assembly. Design modification becomes a matter of changing certain distributed properties. This propagation of information first occurs from skeleton to skeleton, and then from skeleton to part until all necessary part- or subassembly-specific references have been distributed. Designers can then work on a small subsystem without retrieving or regenerating the entire top-level assembly.

This distribution lets designers reference the same information instead of recreating it for each subassembly. Creo Parametric TOOLKIT Design Manager handles the assembly structure with functions for assembly components, features, and copy geometry features.

Continued Population of the Assembly

Populate the assembly with detailed parts and subassemblies in one of two ways:

- Create new components in the context of the assembly
- Model components individually and then bring them into the assembly

Relate individual parts to each other with assembly relations, skeleton models, layouts, and merge features. Creo Parametric TOOLKIT Design Manager functions manage the assembly structure with functions for assembly components, features, and copy geometry features.

Managing Part Interdependencies

Associativity allows you to modify design intent to cause automatic updating of the appropriate objects in your assembly. Associativity is accomplished through external relationships, also known as dependencies or references.

Part interdependencies allow for communication of design criteria from components on one level of the design to components on lower levels. Associativity and part dependencies provide a means for controlled changing or updating of an entire assembly design. Reference control manages part interdependencies by limiting undesirable ones or allowing desirable ones.

External references are dependencies between a Creo Parametric object (part or subassembly) and information from another object not owned by either the referencing object or its submodels. References to “out-of-model” information are external references. Design Manager handles these references with external reference control functions.

Scope is the range of objects to which a specified object can refer. Scope control functions allow you to define objects to which other objects under development can refer. You can establish global scope settings for all objects or specific settings for individual objects.

Design Manager handles scope issues with external reference control functions. The enumerated type `ProExtRefScope` defines possible scope settings as:

- None—Allows no external references.
- Subassembly—Allows external references only to components of the same subassembly
- Skeleton Model—Allows external references to higher-level skeleton models only
- All—Allows external references to any other object anywhere in the assembly

The enumerated type `ProInvalidRefBehavior` defines two methods of handling out-of-scope references. They are as follows:

- Prohibit Out-of-Scope references—Creo Parametric TOOLKIT reports the external reference as out of scope. You must select another reference.
- Copy Out-of-Scope Reference—Creo Parametric TOOLKIT warns that the reference is out of scope. You must do one of the following:
  - Cancel the selection and choose a different reference
  - Confirm that you do want to use the selected reference. Creo Parametric TOOLKIT then creates a “local backup” of the reference. The local backup reference automatically updates (only while the parent is in the current session).

Skeleton Model Functions

Functions Introduced:

- **ProAsmSkeletonMdlnameCreate()**
- **ProAsmSkeletonAdd()**
- **ProAsmSkeletonGet()**
- **ProAsmSkeletonDelete()**
- **ProMdlIsSkeleton()**

Create skeleton models using function `ProAsmSkeletonMdlnameCreate()`. This function creates a new skeleton model with the specified name, adds it to the specified assembly, and initializes the model handle. The input arguments are assembly handle, the skeleton model name, and the handle to the part or skeleton used as a template. If the template handle is `NULL`, an empty skeleton model is created.

By default, the absolute accuracy template is used to create a new model. After you create a new model using the initial value of absolute accuracy, the baseline of the outline is used to determine whether the absolute accuracy is still valid. The outline of the model is calculated after the creation of the first feature of the model or placing the first component in an assembly model. Refer to the Creo Parametric help for more information on Model Accuracy.

`ProAsmSkeletonAdd()` adds an existing skeleton model to the specified assembly. The input arguments are a handle for the assembly to which the skeleton model will be added, and a handle to the skeleton model.

`ProAsmSkeletonGet()` returns a skeleton model of the specified assembly that is currently in memory, then initializes the model handle. The input argument is a handle to the specified assembly.

`ProAsmSkeletonDelete()` deletes a skeleton model component from the specified assembly. The input argument is a handle to the specified assembly.

`ProMdlIsSkeleton()` determines if the specified model is a skeleton model. The input argument is a handle to the model to be checked.

Assembly Component Functions

Functions Introduced:

- **ProAsmcompMdlnameCreateCopy()**
- **ProAsmcompIsUnplaced()**
- **ProAsmcompFillFromMdl()**

Create new components in the specified assembly by copying them from a specified model using `ProAsmcompMdlnameCreateCopy()`. This function creates a new component with the specified name, places it at a default location in the assembly, or leaves it unplaced. The input arguments are the assembly to copy from, the new component name, the new component type (either `PRO_MDL_ASSEMBLY` or `PRO_MODEL_PART`), the handle to the model used as a template, and specification of default or “unplaced” component placement. If the template handle is `NULL`, the component is created empty.

`ProAsmCompIsUnplaced()` determines whether the specified component is unplaced. The input argument is a handle to the component to be checked.

`ProAsmCompFillFromMdl()` copies the specified template model into a model of the specified component. The input arguments are the handle to the component, and the handle to the model used as a template for the copy.

External Reference Control Functions

Functions Introduced:

- **ProRefCtrlSolidSet()**
- **ProRefCtrlSolidGet()**
- **ProRefCtrlEnvirSet()**
- **ProRefCtrlEnvirGet()**
- **ProRefCtrlCheckScope()**

Function `ProRefCtrlSolidSet()` sets a specified external reference control setting on a solid, that is, on a part or assembly. Use `ProRefCtrlSolidGet()` to retrieve the external reference control setting for a specified solid.

`ProRefCtrlEnvirSet()` establishes the run-time environment setting for external reference control. Function `ProRefCtrlEnvirGet()` retrieves this data.

Function `ProRefCtrlCheckScope()` checks whether object-specific reference control settings for a specified model (either an independent object or an assembly component) allow that model to reference information belonging to a different model. The top-level assembly for the component being modified and for the component being referenced must be the same.

If `ProRefCtrlCheckScope()` finds that the owner of the component being modified is `NULL` and the solid (part or assembly) being referenced is not a sub-model of the solid being modified, it reports the reference as out of assembly context. If the `ProMdl` returned is `NULL` but there is a scope violation, the environment scope has been violated.

The enumerated type `ProExtRefScope` defines allowed scope settings for external references as follows:

```
typedef enum{
  PRO_REFCTRL_ALLOW_ALL           = 0, /* all external references allowed*/
  PRO_REFCTRL_ALLOW_SUBASSEMBLY   = 1, /* allow only external references
                                          inside the same higher level
                                          subassembly as that
                                          of the modified object         */
  PRO_REFCTRL_ALLOW_SKELETON      = 2, /* only external references to
                                          skeleton models allowed        */
  PRO_REFCTRL_ALLOW_NONE          = 3  /* no external references allowed */
} ProExtRefScope;
```

Enumerated type `ProInvalidRefBehavior` defines the supported methods for handling Out-of-Scope external references as follows:

```
typedef enum
{  PRO_REFCTRL_BACKUP_REF       = 0,   /* create a local backup for
                                          out-of-scope references        */
  PRO_REFCTRL_PROHIBIT_REF      = 1    /* prohibit out-of-scope external
                                          references                     */
} ProInvalidRefBehavior;
```

Feature and CopyGeom Feature Functions

Functions Introduced:

- **ProFeatureCopiedRefStateDetailsGet()**
- **ProFeatureHasBackup()**
- **ProFeatureDSFDependencystateSet()**
- **ProFeatureDSFDependencystateGet()**
- **ProFeatureDSFDependencyNotifySet()**
- **ProFeatureDSFDependencyNotifyGet()**

The function `ProFeatureCopiedRefStateDetailsGet()` retrieves the status of copied references for a specified feature. This function supports both CopyGeom features and features with local backup of references.

Note

CopyGeom features have no local backup of reference data.

The enumerated type `ProCopiedRefStateDetails` defines possible states of local copies of external references in a CopyGeom feature or in a feature with a local backup.

`ProFeatureHasBackup()` determines if the specified feature has local backup of external references.

`ProFeatureCopyGeomDependSet()` sets copied references of the specified CopyGeom feature to be dependent on the referenced or master model. This means the specified CopyGeom feature references will update as the referenced or master model changes. The function `ProFeatureCopyGeomDependSet()` has been deprecated .Use the function `ProFeatureDSFDependencystateSet()` with dependency status `PRO_DSF_UPDATE_AUTOMATICALLY` instead . The function `ProFeatureCopyGeomInDependSet()` sets copied references to be independent of the referenced or master model, that is, they do not update as this top-level model changes. The function `ProFeatureCopyGeomInDependSet()` has been deprecated .Use the function `ProFeatureDSFDependencystateSet()` with dependency status `PRO_DSF_UPDATE_MANUALLY` instead.

The function `ProFeatureDSFDependencystateSet()` sets the dependency status of data sharing feature(DSF). The dependency status governs the behavior of the data sharing feature if the items referenced by the feature have changed. The valid values for the dependency status are:

- `PRO_DSF_UPDATE_AUTOMATICALLY`—Specifies that all the changes made to the parent model are automatically reflected in the DSF feature.
- `PRO_DSF_UPDATE_MANUALLY`—Specifies that the feature must be updated manually to keep it up-to-date with the referenced model.
- `PRO_DSF_NO_DEPENDENCY`—Specifies that there will be no dependency between the DSF feature and referenced model. This includes updating the geometry of the DSF, automatic retrieval of the referenced model and checking in the model to Windchill.

For more information on the dependency statuses, refer to the section Feature Element Tree on page 1211 in Assembly: Data Sharing Features.

The function `ProFeatureDSFDependencystateGet()` gets the current dependency status for the DSF feature.

Use the function `ProFeatureDSFDependencyNotifySet()` to set the notification status, to visually indicate the changes applied to the source geometry of the data sharing feature.

Use the function `ProFeatureDSFDependencyNotifyGet()` to get a visual indication of the current notification status of the DSF feature. If the notification status is set to on, then any change made in the source geometry is indicated in the form of a yellow triangle in the model tree of the DSF feature in the Creo Parametric user interface. Also, a general notification icon appears, adjacent to the regeneration icon on the status bar in the Creo Parametric user interface. This icon indicates the change in the source geometry of the DSF feature.

Note

Use the functions `ProFeatureDSFDependencyNotifySet()` and `ProFeatureDSFDependencyNotifyGet()` only if the dependency status of the DSF feature is set to `PRO_DSF_UPDATE_MANUALLY`.

External Reference Data Gathering

An external reference or an external dependency is a relationship between an object, such as, a part or subassembly and some information from another object that is not inherently available to the referencing object all the time. While investigating object dependencies in an assembly, some features may exist that

were created in the context of another assembly. All such dependencies are called external dependencies and they point to a component in another assembly. See the Creo Parametric help for more information on external references.

Functions Introduced:

- **ProFeatureExternChildrenGet()**
- **ProFeatureExternParentsGet()**
- **ProSolidExternChildrenGet()**
- **ProSolidExternParentsGet()**
- **ProExtRefInfoFree()**
- **ProExtRefStateGet()**
- **ProExtRefTypeGet()**
- **ProExtRefAsmcompsGet()**
- **ProExtRefOwnMdlGet()**
- **ProExtRefMdlGet()**
- **ProExtRefOwnFeatGet()**
- **ProExtRefFeatGet()**
- **ProExtRefModelitemGet()**
- **ProExtRefInfoExport()**
- **ProExtRefIsDependency()**
- **ProExtRefDependencyIsBreakable()**
- **ProExtRefBreakDependency()**

The function `ProFeatureExternChildrenGet()` retrieves information about external and local children of the specified feature according to the specified reference type. The function `ProFeatureExternParentsGet()` does the same for parents of the feature.

The function `ProSolidExternChildrenGet()` retrieves external and local children of the specified solid according to the specified reference type. The function `ProSolidExternParentsGet()` does the same for parents of the solid.

The function `ProExtRefInfoFree()` releases memory allocated to the external reference data for a feature or solid.

The function `ProExtRefStateGet()` returns the external reference status of the referenced item of the specified reference.

The enumerated type `ProRefState` defines the possible states of top-level solids, such as, part, assembly or component, to which a lower-level solid refers.

The function `ProExtRefTypeGet()` returns the type of the external reference.

The enumerated type `ProExtRefType` defines the supported external reference types as follows:

```
typedef enum
{
  PRO_EXT_GEOM_REF    = 1,       /* all out of solid references,
                                    created in assembly context, kept in
                                    plins, sections, draft sections        */
  PRO_LOC_GEOM_REF    = 2,       /* local for solid references, kept in
                                    plins, sections, draft sections        */
  PRO_MERGE_REF       = 3,       /* reference models of merge by ref feats  */
  PRO_EXT_REL_REF     = 4,       /* out of solid references, , kept in
                                    symbols used for relations.
                                    Can be "to solid" or feature,
                                    geometry references.                   */
  PRO_LOC_REL_REF     = 5,       /* local for solid references, kept in
                                    symbols used for relations.
                                     Can be "to solid" or feature,
                                     geometry references.                   */
  PRO_PRGM_REF        = 6,       /* out of solid references, , kept in
                                    symbols used in Pro/Program.
                                    Always solid references                */
  PRO_MOVE_COMP_REF   = 7,       /* Move Components external references.
                                   Kept in components and always "to solid".
 This reference is not present in models
 created after Creo Elements/Pro 5.0    */
  PRO_SUBS_REF        = 8,       /* Substitute Component references.
                                  Kept in components and always "to solid"*/
  PRO_MFG_INFO_REF    = 9,       /* Mfg Info references.  Kept in
                                  mfg feat, always "to solid"              */
  PRO_INTRCH_REF      = 10,      /* Interchange Assembly references.
                               Kept in the solid itself.
                               Always "to solid"                       */
  PRO_HARN_REF        = 11,      /* Harness references.
                                 Kept in the solid itself.
                                Always "to solid"                         */
  PRO_FEAT_PAT_REF    = 12,      /* Feature pattern references.
                                  Does not include pattern relation
                                 references.  Always "to solid"          */
  PRO_NON_ASSY_GEOM_REF = 13,    /* Out of solid external geometry refs,
                                  created not in assembly context,
                                 kept in plins. (used in external geom
                                  copy feature).                          */
  PRO_DIM_BOUND_REF     = 14,    /* Dim. bound references.
                                  Kept in the solid itself.
                                 Always "to solid"                       */
  PRO_HIDDEN_FEM_REFS   = 15,    /* Hidden Simulate features references.
                                  Kept in the solid itself.
                                  Always "to solid"                       */
  PRO_ANALYSIS_REF      = 16,    /* Hidden analysis features references.
                                  Kept in the solid itself.
                                  Always "to solid"                       */
```

```
  PRO_FEAT_PAT_LOC_REF  = 17,    /* References between pat. leader and member
                                    or between pat. group headers          */
  PRO_DEPENDENCY_REFS   = 18,    /* All types of references collected via
                                    collect dependencies mechanism. This type
is not included in COLL_ALL_REFS_TYPE, it
should be invoked separately.            */
  PRO_IN_CIRCLE_REFS    = 19,    /* References encountered in assembly loops.
This is reserved for future use */
  PRO_MEMBER_REFS       = 20,    /* Component models of assembly members.
This type is not included in
COLL_ALL_REFS_TYPE, it should be invoked
separately.  */
  PRO_LOC_MERGE_REF     = 21,    /* Merge reference of mirror geom.
                                    Always "to solid"                      */
  PRO_ALL_EXT_REF_TYPES = 100,   /* Same as PRO_ALL_REF_TYPES except for
PRO_LOC_GEOM_REF, PRO_LOC_REL_REF,
PRO_LOC_MERGE_REF, PRO_FEAT_PAT_LOC_REF  */
  PRO_ALL_REF_TYPES     = 101    /* All known types of references, except for
PRO_DEPENDENCY_REFS,PRO_IN_CIRCLE_REFS and
PRO_MEMBER_REFS.              */
} ProExtRefType;                 /* types of references */
```

The structures `ProExtFeatRef` and `ProExtRefInfo` provide pointers to a structure containing external references for a specified feature:

```
typedef struct ext_feat_ref *ProExtFeatRef;
typedef struct
{
   ProExtRefType    type;   ProExtFeatRef   *ext_refs;
   int              n_refs;
} ProExtRefInfo;
```

The function `ProExtRefAsmcompsGet()` retrieves from the specified external reference a path to the component from which the reference was created. It also returns a path to the component that owns the specified external reference.

The function `ProExtRefOwnMdlGet()` retrieves a solid that is active in the session and uses the provided reference . The function `ProExtRefMdlGet()` retrieves a solid, in a model that is active in the session. This returned solid is referred to by the specified external reference.

The function `ProExtRefOwnFeatGet()` retrieves from the specified external reference a feature that uses the reference. The function `ProExtRefFeatGet()` retrieves from the specified external reference a feature referred to by the external reference.

The function `ProExtRefModelitemGet()` retrieves from the specified external reference a model item that uses that reference.

The function `ProExtRefInfoExport()` prints out a dependency report for all references of type `PRO_DEPENDENCY_REFS` in the specified format. The input arguments of this function are:

- `info_arr`—Specify all the references of the type `PRO_DEPENDENCY_REFS` collected using the functions `ProSolidExternParentsGet()` and `ProFeatureExternParentsGet()`. All the references that are not dependencies will be ignored.
- `w_fname`—Specify the name of the file to which the report is to be printed.
- `n_rep_type`—Specify the type of report format. The valid values for this input argument are:
  - `PRO_REPORT_TYPE_CSV`—Specifies a comma separated value file.
  - `PRO_REPORT_TYPE_XML`—Specifies a XML file.

The function `ProExtRefIsDependency()` indicates if the specified reference is an external dependency.

The function `ProExtRefDependencyIsBreakable()` indicates if some of the specified dependencies can be broken or not, in case the corresponding external references are not required.

The function `ProExtRefBreakDependency()` breaks the external references from the specified array of references. Among all references in the specified array, this acts only on those external references that are breakable. As a result of break operation, the dependency associated with the external reference is broken, which prevents the formation of ghost objects in Product Development Management System. Refer the Creo Parametric Help for more information on breaking dependencies.

55

Assembly: Assembling Components

This chapter describes how to use the concepts of feature creation to assemble components into an assembly. Read the chapter Element Trees: Principles of Feature Creation on page 764 before this chapter.

Assembling Components by Functions

Functions Introduced:

- **ProAsmcompMdlnameCreateCopy()**
- **ProAsmcompAssemble()**
- **ProAsmcompPositionGet()**
- **ProAsmcompPositionSet()**
- **ProAsmcompConstraintsWithComppathGet()**
- **ProAsmcompConstraintsSet()**
- **ProAsmcompAllConstrRemove()**
- **ProAsmcompConstrRemove()**
- **ProAsmcompRegenerate()**

Superseded Function:

- **ProAsmcompConstraintsWithDtmOrientGet()**

Use the function `ProAsmcompMdlnameCreateCopy()` to create a new component in the assembly by copying from an existing model. Specify the handle to the model to be used as a template for the copy. If a model is not specified, a component that does not have initial geometry is created. The function provides the `ProAsmcomp` handle to the new component.

The function `ProAsmcompAssemble()` assembles a component to the assembly or sub-assembly using the parametric constraints available when assembling a component in Creo Parametric. The initial position of the component is a `ProMatrix` object. Specify the orientation of the three axes and the position of the origin of the component coordinate system, with respect to the target assembly coordinate system. The function provides the `ProAsmcomp` feature handle to the newly created assembly.

Note

If the transform matrix passed as the initial position of the component is incorrect and non-orthonormal, the function `ProAsmcompAssemble()` returns the error `PRO_TK_BAD_INPUTS`. In such scenario, you can use the function `ProMatrixMakeOrthonormal()` to convert this non-orthonormal matrix to an orthonormal matrix.

The function `ProAsmcompPositionGet()` retrieves the component's initial position before constraint are applied.

The function `ProAsmcompPositionSet()` specifies the initial position of the component before constraints are applied. This affects the position of the component only if the component is packaged or underconstrained.

The function `ProAsmcompConstraintsWithDtmOrientGet()` is deprecated in Creo Parametric 7.0.0.0. Use the function `ProAsmcompConstraintsWithCompppathGet()` instead. The function retrieves the specified constraints for the given assembly component and component path. The orientation of the constraints is returned as a value of enumerated data type `ProDatumside`. The input argument *component_path* is the path to the owner assembly only if the constraints have references to the other members of the top-level assembly. If the constraints have references only to the owner assembly, then you need to pass this as `Null`.

The function `ProAsmcompConstraintsSet()` sets an array of constraints for a given assembly component. This function modifies the component feature data and regenerates the assembly component.

The function `ProAsmcompAllConstrRemove()` removes all types of constraints including interface constraints for the specified assembly component. Specify a a `ProAsmcomp` handle to the assembly component in the input argument *p_feat_handle*.

The function `ProAsmcompConstrRemove()` removes one or all constraints for the specified assembly component. However, the function does not remove the interface constraint. The input arguments are as follows:

- *p_feat_handle*—Specifies a `ProAsmcomp` handle to the assembly component.
- *index*—Specifies the constraint index. Pass the value as `-1` to remove all the constraints. Use the function `ProAsmcompConstraintGet()` to determine the index of a particular constraint.

The function `ProAsmcompRegenerate()` regenerates the placement instructions for an assembly component, given the component handle. The function regenerates the placement instructions just as in an interactive Creo Parametric session. Alternatively, you can use the visit functionality to regenerate recursively some or all of the components in the assembly.

Assembling a Component Parametrically

Functions Introduced:

- **ProAsmcompconstraintAlloc()**
- **ProAsmcompconstraintTypeGet()**
- **ProAsmcompconstraintTypeSet()**
- **ProAsmcompconstraintAsmreferenceGet()**

- **ProAsmcompconstraintAsmreferenceSet()**
- **ProAsmcompconstraintCompreferenceGet()**
- **ProAsmcompconstraintCompreferenceSet()**
- **ProAsmcompconstraintOffsetGet()**
- **ProAsmcompconstraintOffsetSet()**
- **ProAsmcompconstraintAttributesGet()**
- **ProAsmcompconstraintAttributesSet()**
- **ProAsmcompconstraintUserdataGet()**
- **ProAsmcompconstraintUserdataSet()**
- **ProAsmcompconstraintFree()**
- **ProAsmcompconstraintArrayFree()**

The function `ProAsmcompconstraintAlloc()` allocates memory for the constraint data structure. This data structure describes the types of constraints that can be applied to the assembly component.

The function `ProAsmcompconstraintTypeGet()` retrieves the constraint type of the specified constraint. The types of constraints are:

- `PRO_ASM_UNDEF`—Use this option to initialize a variable. This option is never returned by the function `ProAsmcompconstraintTypeGet()` and can be ignored.
- `PRO_ASM_MATE`—Use this option to make two surfaces coincident with one another and facing each other.
- `PRO_ASM_MATE_OFF`—Use this option to make two planar surfaces parallel and facing each other.
- `PRO_ASM_ALIGN`—Use this option to make two planes coplanar, two axes coaxial or two points coincident. You can also align revolved surfaces or edges.
- `PRO_ASM_ALIGN_OFF`—Use this option to align two planar surfaces at an offset.
- `PRO_ASM_INSERT`—Use this option to insert a "male" revolved surface into a "female" revolved surface, making their respective axes coaxial.
- `PRO_ASM_ORIENT`—Use this option to make two planar surfaces to be parallel in the same direction.
- `PRO_ASM_CSYS`—Use this option to place a component in an assembly by aligning the coordinate system of the component with the coordinate system of the assembly.
- `PRO_ASM_TANGENT`—Use this option to force two surfaces to be tangent.

- PRO_ASM_PNT_ON_SRF—Use this option to align a point with a of a surface.
- PROS_ASM_EDGE_ON_SRF—Use this option to align a straight edge with a surface.
- PRO_ASM_DEF_PLACEMENT—Use this option to align the default coordinate system of the component to the default coordinate system of the assembly.
- PRO_ASM_SUBSTITUTE—This constraint type is used in simplified representations only when the component is replaced by a substitute component.
- PRO_ASM_PNT_ON_LINE—Use this option to force the intersection of a line with a point.
- PRO_ASM_FIX—Use this option to fix the current location of the component as a constraint.
- PRO_ASM_AUTO—Not for use by Creo Parametric TOOLKIT.
- PRO_ASM_ALIGN_ANG_OFF—This option can only be used in conjunction with another constraint. If you have two flat surfaces and create an align edge or axis constraint where the edge or axis lies on the surface, then you can specify an angle offset constraint between the two surfaces.
- PRO_ASM_MATE_ANG_OFF—This option can only be used in conjunction with another constraint. If you have two flat surfaces and create a mate edge or axis constraint where the edge or axis lies on the surface, then you can specify an angle offset constraint between the two surfaces.
- PRO_ASM_CSYS_PNT—This option can be used in **User Defined**, **General**, and **Gimbal** connections. Use this option to place a component in an assembly by aligning the origins of the coordinate systems. Here the axes are not aligned, and thus, the component can be freely rotated along the three rotation axes. In **User Defined** and **Rigid** connections you can switch from **Coord Sys Point** to **Coord Sys** constraint and vice-versa.

Use the function ProAsmcompconstraintTypeSet() to set the constraints for the assembly component constraint.

The function ProAsmcompconstraintAsmreferenceGet() retrieves the ProSelection handle to a reference on the assembly and the orientation of the assembly for the specified assembly component constraint. The assembly orientation can have the following values:

- PRO_DATUM_SIDE_YELLOW—The primary side of the datum plane which is the default direction of the arrow.
- PRO_DATUM_SIDE_RED—The secondary side of the datum plane which is the direction opposite to that of the arrow.
- PRO_DATUM_SIDE_NONE—No orientation is specified.

The assembly orientation is applicable for legacy models prior to M260.

The function ProAsmcompconstraintAsmreferenceSet() selects a reference on the assembly and sets the orientation of the assembly for a specified assembly component constraint.

Note

The assembly reference selection must be assigned an assembly component path, even if the reference geometry is in the top-level assembly. In that situation the table_num value of the ProAsmcomppath structure would be 0.

The function ProAsmcompconstraintCompreferenceGet() retrieves the ProSelection handle to a reference on the placed component and the orientation of the component for the specified assembly component constraint. The component orientation can have the following values:

- PRO_DATUM_SIDE_YELLOW—The primary side of the datum plane which is the default direction of the arrow.
- PRO_DATUM_SIDE_RED—The secondary side of the datum plane which is the direction opposite to that of the arrow.
- PRO_DATUM_SIDE_NONE—No orientation is specified.

The component orientation is applicable for legacy models prior to M260.

The function ProAsmcompconstraintCompreferenceSet() selects a reference on the placed component and sets the orientation of the component for a specified assembly component constraint.

ProAsmcompconstraintOffsetGet() retrieves the offset value from the reference for the Mate or Align constraint type and the function ProAsmcompconstraintOffsetSet() defines the offset value.

The function ProAsmcompconstraintAttributesGet() retrieves the constraint attributes for the specified assembly component constraint. The function ProAsmcompconstraintAttributesSet() sets the constraint attributes. The types of constraint attributes are:

- PRO_ASM_CONSTR_ATTR_FORCE—Force the constraint, causing strict alignment for axes, lines, and points. You can force a constraint only if the constraint type is Align.
- PRO_ASM_CONSTR_ATTR_IGNORE—Not for use by Creo Parametric TOOLKIT.
- PRO_ASM_CONSTR_ATTR_NONE—No constraint attributes are specified. This is the default value.
- PRO_ASM_CONSTR_ATTR_INTFC_DEPENDENT—When set in a component interface, the constraint cannot be changed by application of settings making it coincident, offset, or reoriented.
- PRO_ASM_CONSTR_ATTR_INACTIVE—The constraint should not be applied to the feature. This corresponds to the **Constraint Enabled** check box in the component feature user interface.

The function ProAsmcompconstraintUserdataGet() retrieves the user data for the given constraint while the function ProAsmcompconstraintUserdataSet() specifies the user data for the given constraint.

Use the function ProAsmcompconstraintFree() to free the constraint data structure from the memory.

The function ProAsmcompconstraintArrayFree() provides a single function to use to free an entire ProArray of ProAsmcompconstraint structures.

Example 1: Component Constraints

The sample code in the file UgAsmcompConstraint.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_asm displays each constraint of the component visually on the screen, and includes a text explanation for each constraint.

Example 2: Assembling Components

The sample code in the file UgAsmcompConstraint.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_asm demonstrates how to assemble a component into an assembly, and how to constrain the component by aligning datum planes. If the complete set of datum planes is not found, the function will show the component constraint dialog to the user to allow them to adjust the placement as they wish.

Redefining Components Interactively

The functions described in this section enable you to reroute previously assembled components, as in an interactive session of Creo Parametric.

Functions Introduced:

- **ProAsmcompConstrRedefUI()**
- **ProAsmcompPackageMove()**

The function `ProAsmcompConstrRedefUI()` is intended for use in interactive Creo Parametric TOOLKIT applications. This function displays the Creo Parametric Component Placement dialog box, enabling you to redefine the constraints interactively. Control is given back to the Creo Parametric TOOLKIT application when you select **OK** or **Cancel** the dialog box is closed.

The function `ProAsmcompPackageMove()` supports interactive and programmatic packaging of components. The arguments to this function allow the user or the program to repackage an existing component. If the component is used for interactive purposes, the control is given back to the Creo Parametric TOOLKIT application when you select **OK** or **Cancel** the dialog box is closed.

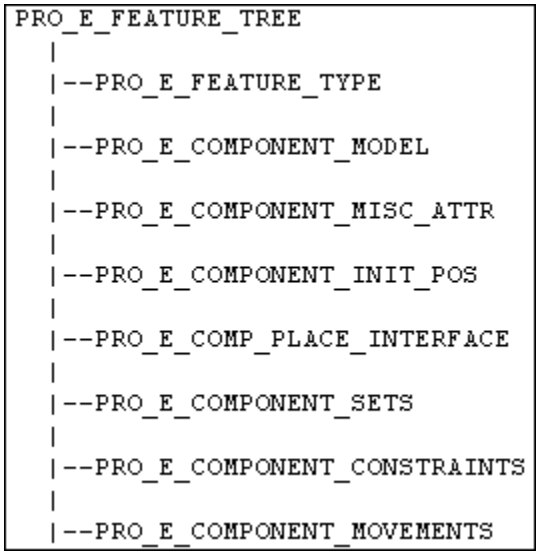
Assembling Components by Element Tree

Assembly components are treated as features in Creo Parametric, so it is logical to replace those dedicated functions by a feature element tree that provides the same functionality, but uses the existing Creo Parametric TOOLKIT functions `ProFeatureCreate()`, `ProFeatureRedefine()` and `ProFeatureElemtreeExtract()`.

The Element Tree for an Assembly Component

The element tree for a component assembly is documented in the header file `ProAsmcomp.h` and is shown in the following figure.

Top-level Feature Element tree for Component Assembly



Model

The element `PRO_E_COMPONENT_MODEL` defines the model to be assembled into the assembly.

Attributes

The element `PRO_E_MISC_ATTR` defines the type of component being assembled. You can also access this property through `ProAsmcompTypeGet()` and `ProAsmcompTypeSet()`.

Initial Position

The element `PRO_E_COMPONENT_INIT_POS` defines the absolute position of the component in the absence of any parametric constraints. Its value is a `ProMatrix` object describing the component position. If you supply this element but no `PRO_E_COMPONENT_CONSTRAINT` elements, the component is assembled as packaged at this location.

If you under constrain the component, the value of this element is used in conjunction with the constraints to set the packaged position for the component.

Constraint Sets and Mechanism Connections

The element `PRO_E_COMPONENT_SETS` is an array of elements of type `PRO_E_COMPONENT_SET`. Each of these elements is a predefined set of constraints.

Although you can define a user-defined type of constraint set containing any type and combination of constraints, the set element is also important when used to create predefined types of sets in Creo Parametric.

Mechanism uses predefined set types to represent connections. The connection can also contain elements describing mechanism motion axes stored for the connection.

PRO_E_COMPONENT_SET

```
PRO_E_COMPONENT_SET
  |--PRO_E_COMPONENT_SET_ID
  |--PRO_E_COMPONENT_SET_TYPE
  |--PRO_E_COMPONENT_SET_NAME
  |--PRO_E_COMPONENT_SET_MISC_ATTR
  |--PRO_E_COMPONENT_SET_FLIP
  |--PRO_E_COMPONENT_JAS_SETS
       |--PRO_E_COMPONENT_JAS_SET
            |--PRO_E_COMPONENT_JAS_ZERO_TYPE
            |--PRO_E_COMPONENT_JAS_REFS
            |     |--PRO_E_COMPONENT_JAS_ORANGE_REF
            |     |--PRO_E_COMPONENT_JAS_GREEN_REF
            |     |--PRO_E_COMPONENT_JAS_0_OFFSET_VAL
            |--PRO_E_COMPONENT_JAS_REGEN_VALUE_GROUP
            |     |--PRO_E_COMPONENT_JAS_REGEN_VALUE_FLAG
            |     |--PRO_E_COMPONENT_JAS_REGEN_VALUE
            |--PRO_E_COMPONENT_JAS_MIN_LIMIT
            |     |--PRO_E_COMPONENT_JAS_MIN_LIMIT_FLAG
            |     |--PRO_E_COMPONENT_JAS_MIN_LIMIT_VAL
            |     |--PRO_E_COMPONENT_JAS_MIN_LIMIT_REF
            |--PRO_E_COMPONENT_JAS_MAX_LIMIT
            |     |--PRO_E_COMPONENT_JAS_MAX_LIMIT_FLAG
            |     |--PRO_E_COMPONENT_JAS_MAX_LIMIT_VAL
            |     |--PRO_E_COMPONENT_JAS_MAX_LIMIT_REF
            |--PRO_E_COMPONENT_JAS_RESTITUTION
            |     |--PRO_E_COMPONENT_JAS_RESTITUTION_FLAG
            |     |--PRO_E_COMPONENT_JAS_RESTITUTION_COEF
            |--PRO_E_COMPONENT_JAS_FRICTION
                  |--PRO_E_COMPONENT_JAS_FRICTION_FLAG
                  |--PRO_E_COMPONENT_JAS_STATIC_FRIC_COEF
                  |--PRO_E_COMPONENT_JAS_KINEM_FRIC_COEF
                  |--PRO_E_COMPONENT_JAS_RADIUS_VALUE
```

Each component set contains the following elements:

- PRO_E_COMPONENT_SET_ID—Specifies the component set id. This value is generated automatically by Creo Parametric upon creation of the set. The set ids should remain the same when redefining the component.
- PRO_E_COMPONENT_SET_TYPE—Specifies the component set type. The following table describes the valid values in details.

Component Set Type	Description
PRO_ASM_SET_TYPE_FIXED	A "Rigid" Mechanism connection: Connects two components so that they do not move relatively to each other.
PRO_ASM_SET_TYPE_PIN	A "Pin" Mechanism connection: Connects a component to a referenced axis so that the component rotates or moves along this axis with one degree of freedom.
PRO_ASM_SET_TYPE_SLIDER	A "Slider" Mechanism connection: Connects a component to a referenced axis so that the component moves along the axis with one degree of freedom.
PRO_ASM_SET_TYPE_CYLINDRICAL	A "Slider" Mechanism connection: Connects a component so that it moves along and rotates about a specific axis with two degrees of freedom.
PRO_ASM_SET_TYPE_PLANAR	A "Planar" Mechanism connection: Connects components so that they move in a plane relatively to each other with two degrees of freedom in the plane and one degree of freedom around an axis perpendicular to it.
PRO_ASM_SET_TYPE_BALL	A "Ball" Mechanism connection: Connects a component so that it can rotate in any direction with three degrees of freedom (360° rotation).
PRO_ASM_SET_TYPE_WELD	A "Weld" Mechanism connection: Connects a component to another so that they do not move relatively to each other.
PRO_ASM_SET_TYPE_BEARING	A "Bearing" Mechanism connection: A combination of Ball and Slider connections with four degrees of freedom.
PRO_ASM_SET_TYPE_GENERAL	A "General" Mechanism connection: Has one or two configurable constraints that are identical to those in a user-defined set.
PRO_ASM_SET_TYPE_6DOF	A "Six Degrees of Freedom" Mechanism connection: Does not affect the motion of the component in relation to the assembly because no constraints are applied.
PRO_ASM_SET_TYPE_GIMBAL	A “Gimbal” Mechanism connection: This connection behaves similar to the "Six Degrees of Freedom" connection except that in Gimbal connection the translational degrees of freedom are locked.

Component Set Type	Description
PRO_ASM_SET_TYPE_SLOT	A "Slot" Mechanism connection: A point on a non straight trajectory. This connection has four degrees of freedom, where the point follows the trajectory in three directions.
PRO_ASM_SET_USER_DEFINED_TYPE	A user defined constraint set. Legacy components which do not have defined Mechanism connections will always use this type.

- PRO_E_COMPONENT_SET_NAME—Specifies the name of the component set.
- PRO_E_COMPONENT_SET_MISC_ATTR—Specifies the component set attributes. Currently, these attributes are limited to flags which enable or disable the set.
- PRO_E_COMPONENT_JAS_SETS is an array of compound elements of type PRO_E_COMPONENT_JAS_SET which indicate Joint Axis Set (JAS). It represents the motion axis settings for the Mechanism connection. It consists of the following elements—
  - PRO_E_COMPONENT_JAS_ZERO_TYPE—Specifies the type of motion represented by this motion axis element and the value is drawn from the following types:
    - PRO_AXIS_ZERO_TRANSLATE1
    - PRO_AXIS_ZERO_TRANSLATE2
    - PRO_AXIS_ZERO_TRANSLATE3
    - PRO_AXIS_ZERO_ROTATION1
    - PRO_AXIS_ZERO_ROTATION2
    - PRO_AXIS_ZERO_ROTATION3
    - PRO_AXIS_ZERO_SLOT
  - PRO_E_COMPONENT_JAS_REFS—Specifies initial position references for the motion axis.
    - PRO_E_COMPONENT_JAS_ORANGE_REF—Specifies the component reference of the motion axis.
    - PRO_E_COMPONENT_JAS_GREEN_REF—Specifies the assembly reference of the motion axis.
    - PRO_E_COMPONENT_JAS_0_OFFSET_VAL—Specifies the zero offset value for the motion axis.
  - PRO_E_COMPONENT_JAS_REGEN_VALUE_GROUP is a compound element specifying the motion axis regeneration options. The motion axis regeneration value is an offset value used when regenerating the model.

This value determines the offset of the component being placed in the assembly from the zero reference during regeneration.

- PRO_E_COMPONENT_JAS_REGEN_VALUE_FLAG—Specifies enabling or disabling the regeneration value. When you disable it, the motion axis becomes free of constraint.
- PRO_E_COMPONENT_JAS_REGEN_VALUE— Specifies the value along this motion axis that will be used for regeneration of the position and orientation of the component.

- PRO_E_COMPONENT_JAS_MIN_LIMIT specifies the minimum limit for regeneration value. It consists of the following elements:
  - PRO_E_COMPONENT_JAS_MIN_LIMIT_FLAG—A boolean element indicating whether the minimum limit is applied to the component.
  - PRO_E_COMPONENT_JAS_MIN_LIMIT_VAL—The value for the minimum limit.
  - PRO_E_COMPONENT_JAS_MIN_LIMIT_REF—A selected item which serves as the minimum limit instead of using an assigned value.
- PRO_E_COMPONENT_JAS_MAX_LIMIT—Specifies the maximum limit for regeneration value. It consists of the following elements:
  - PRO_E_COMPONENT_JAS_MAX_LIMIT_FLAG—A boolean element indicating if the maximum limit is applied to the component.
  - PRO_E_COMPONENT_JAS_MAX_LIMIT_VAL—The value for the maximum limit.
  - PRO_E_COMPONENT_JAS_MAX_LIMIT_REF—A selected item, which serves as the maximum limit instead of using an assigned value.
- PRO_E_COMPONENT_JAS_RESTITUTION—Specifies coefficient of restitution for the motion axis. It is a compound element and consists of the following elements:
  - PRO_E_COMPONENT_JAS_RESTITUTION_FLAG—A boolean element indicating if a coefficient of restitution is applied.
  - PRO_E_COMPONENT_JAS_RESTITUTION_COEF— Specifies the coefficient of restitution value.
- PRO_E_COMPONENT_JAS_FRICTION—Specifies coefficient of friction for the motion axis. It is a compound element and consists of the following elements:
  - PRO_E_COMPONENT_JAS_FRICTION_FLAG—A boolean element indicating if the coefficients of friction are applied.

- PRO_E_COMPONENT_JAS_STATIC_FRIC_COEF— Specifies the coefficient of static friction value.
- PRO_E_COMPONENT_JAS_KINEM_FRIC_COEF— Specifies the coefficient of kinetic friction value.
- PRO_E_COMPONENT_JAS_RADIUS_VALUE— Specifies the contact radius value.

Placement Constraints

The element PRO_E_COMPONENT_CONSTRAINTS is an array of elements of type PRO_E_COMPONENT_CONSTRAINT, each representing a single component placement constraint.

PRO_E_COMPONENT_CONSTRAINT

```
PRO_E_COMPONENT_CONSTRAINT
  |--PRO_E_COMPONENT_CONSTR_TYPE
  |--PRO_E_COMPONENT_COMP_CONSTR_REF
  |--PRO_E_COMPONENT_ASSEM_CONSTR_REF
  |--PRO_E_COMPONENT_CONSTR_REF_OFFSET
  |--PRO_E_COMPONENT_USER_DATA
  |--PRO_E_COMPONENT_CONSTR_ATTR
  |--PRO_E_COMPONENT_COMP_ORIENT
  |--PRO_E_COMPONENT_ASSM_ORIENT
  |--PRO_E_COMPONENT_CONSTR_SET_ID
  |--PRO_E_COMPONENT_SLOT_EXTRA_CRV_REF
```

Each constraint element contains the following elements:

- PRO_E_COMPONENT_CONSTR_TYPE—See discussion regarding Assembling a Component Parametrically on page 1161.
- PRO_E_COMPONENT_COMP_CONSTR_REF—Identifies the geometry item in the component referenced by the constraint. This element is of type Selection.
- PRO_E_COMPONENT_ASSEM_CONSTR_REF—Identifies the constraint reference in the assembly.

> **Note**
>
> this reference must include a component path referencing the top level assembly, even if the reference belongs directly to the top level assembly.

- PRO_E_COMPONENT_CONSTR_REF_OFFSET—Gives the offset value, if the constraint type is an offset.

- PRO_E_COMPONENT_USER_DATA—Specifies user data.
- PRO_E_COMPONENT_CONSTR_ATTR—See discussion under Assembling a Component Parametrically on page 1161.
- The elements PRO_E_COMPONENT_COMP_ORIENT and PRO_E_COMPONENT_ASSM_ORIENT indicate which side of a referenced surface to be used. These values have different meanings for user-defined constraint sets and mechanism connections. See discussion under ProAsmcompconstraintAsmreferenceGet() and ProAsmcompconstraintCompreferenceGet().
- PRO_E_COMPONENT_CONSTR_SET_ID—Specifies the index of the member of the array of PRO_E_COMPONENT_SET elements that owns the constraint.
- PRO_E_COMPONENT_SLOT_EXTRA_CRV_REF—Specifies the extra curve references used by a Slot connection only. This is a multivalued element.

Component Movement in Assembly

The element PRO_E_COMPONENT_MOVEMENTS is an array of elements of type PRO_E_COMPONENT_MOVEMENT which represent movements applied to the component being assembled.

PRO_E_COMPONENT_MOVEMENT

```
PRO_E_COMPONENT_MOVEMENT
  |--PRO_E_COMPONENT_MOVEMENT_TYPE
  |--PRO_E_COMPONENT_MOVEMENT_REF
  |--PRO_E_COMPONENT_MOVEMENT_VALUE
```

Each movement element contains the following elements:

- PRO_E_COMPONENT_MOVEMENT_TYPE specifies allowed movement types

Use the following options to move the component parallel to the reference selected and are of the following values:

- PRO_ASM_TRANSLATE_X
- PRO_ASM_TRANSLATE_Y
- PRO_ASM_TRANSLATE_Z

Use the following options to rotate the component about the selected references.

- PRO_ASM_ROTATE_X
- PRO_ASM_ROTATE_Y
- PRO_ASM_ROTATE_Z
- PRO_ASM_TWIST_FIT

- PRO_E_COMPONENT_MOVEMENT_REF specifies the translational and rotational motion references.
- PRO_E_COMPONENT_MOVEMENT_VALUE

Placement via Interface

The element PRO_E_COMP_PLACE_INTERFACE is a compound element that defines an alternative assembly technique: using component interfaces to define the placement instead of traditional constraints.

PRO_E_COMP_PLACE_INTERFACE

```
PRO_E_COMP_PLACE_INTERFACE
  |--PRO_E_COMP_PLACE_INTERFACE_TYPE
  |--PRO_E_COMP_PLACE_INTERFACE_COMP
  |--PRO_E_COMP_PLACE_INTERFACE_ASSEMS
       |--PRO_E_COMP_PLACE_INTERFACE_ASSEM
            |--PRO_E_COMP_PLACE_INTERFACE_ASSEM_REF
```

The element contains the following elements:

- PRO_E_COMP_PLACE_INTERFACE_TYPE—Specifies the interface types as follows:
  - PRO_ASM_INTFC_TO_GEOM—Assembly of the component by matching an interface on the component to referenced geometry in the assembly.
  - PRO_ASM_INTFC_TO_INTFC—Assembly of the component by matching an interface on the component to an interface defined in the assembly.

> **Note**
>
> If this value is not set (set to 0) then component interfaces are not used to define this component.

- PRO_E_COMP_PLACE_INTERFACE_COMP—Specifies the component model interface. This should contain the component interface feature.
- PRO_E_COMP_PLACE_INTERFACE_ASSEMS—Specifies an array of assembly references. If the placement type is PRO_ASM_INTFC_TO_GEOM this contains 1 or more geometric references from the assembly. If the placement type is PRO_ASM_INTFC_TO_INTFC this contains a single reference element containing the component interface feature.

Assembling Components Using Intent Datums

You can use Intent Datums such as Intent Axis, Intent Point, Intent Plane, and Intent Coordinate System in assembly component placement and constraints.

Example 3: Assembling Components Using Intent Datums

The sample code in the file `UgAsmcompConstraint.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_asm` demonstrates how to assemble a component into an assembly using intent point and intent plane, and how to set constraints after assembling a component using intents.

56

Assembly: Kinematic Dragging and Creating Snapshots

This chapter describes the Creo Parametric TOOLKIT functions for dragging assembly entities through an allowable range to see how your assembly works in a specific configuration. These functions provide the same result as obtained by using the **Drag** dialog box in the Creo Parametric user interface.

This chapter also describes the functions to create snapshots of the assembly in different positions and orientations.

Connecting to a Kinematic Drag Session

Functions Introduced:

- **ProKinDragStart()**
- **ProKinDragStop()**
- **ProKinDragSessionInquire()**
- **ProKinDragEnvironmentSet()**
- **ProKinDragReconnect()**

The function `ProKinDragStart()` starts a drag session. This function should be called before any other function for Kinematic drag or snapshots. All subsequent functions work with the snapshots of the top-level assembly. This function can be called for design assemblies in the Standard Assembly and Mechanism modes only. It cannot be used in the following cases:

- If the assembly is in the exploded state
- If the Sketcher mode is active
- When component placement is active
- When the **Drag** dialog box is active

The function `ProKinDragStop()` stops the drag session. Use `ProWindowRefresh()` to view the change in the graphic window scale according to the updated outline.

The function `ProKinDragSessionInquire()` determines if the drag session is active. The drag session may exist automatically in some cases, such as redefine, regeneration, window switch, representation mode change.

The function `ProKinDragEnvironmentSet()` sets the environment for dragging and reconnect operations. It requires the following input arguments:

- *snap_name*—Specifies the name of the active snapshot whose constraints are used for the drag operation. The snapshot is checked for statuses such as `good`, `outdated`, or `incomplete`. Pass `NULL` if not required.

  The `outdated` status means some parts from the body have a relative transformation that is different from the current transformation. The `incomplete` status means that some parts from the body are missing in the active snapshot.
- *path*—Specifies the path in terms of the `ProAsmcomppath` object to the active snapshot of a subassembly contained within the top-level assembly. Pass `NULL` for a top-level assembly snapshot.
- *type*—Specifies the type of drag to be performed. It is given by the enumerated type `ProKinDragType` that takes the following values:

- PRO_KIN_POINT_DRAG— Select a point to drag in a part within the top-level assembly. During the dragging operation, the point you selected follows the pointer's movement while maintaining connections.
- PRO_KIN_BODY_DRAG—Select a part from the top-level assembly to drag. When you drag a part, its position in the graphics window changes, but its orientation remains fixed.If the assembly requires a part to be reoriented in conjunction with a change in position, the part does not move at all, as the assembly cannot reassemble in the new position.
- PRO_KIN_ADVANCED_TRANS_X—Specifies the translation in the X direction of the selected coordinate system. This type is applicable only in case of an advanced drag operation. You can select a coordinate system by selecting the part to be dragged.
- PRO_KIN_ADVANCED_TRANS_Y—Specifies the translation in the Y direction of the selected coordinate system. This type is applicable only in case of an advanced drag operation.
- PRO_KIN_ADVANCED_TRANS_Z—Specifies the translation in the Z direction of the selected coordinate system. This type is applicable only in case of an advanced drag operation.
- PRO_KIN_ADVANCED_ROT_X—Specifies the rotation around the X axis of the selected coordinate system. This type is applicable only in case of an advanced drag operation. You can select a coordinate system by selecting the part to be dragged.
- PRO_KIN_ADVANCED_ROT_Y—Specifies the rotation around the Y axis of the selected coordinate system. This type is applicable only in case of an advanced drag operation.
- PRO_KIN_ADVANCED_ROT_Z—Specifies the rotation around the Z axis of the selected coordinate system. This type is applicable only in case of an advanced drag operation.

- *refobject*—Specifies the selection reference in the form a coordinate system (PRO_CSYS), part (PRO_PART), or mechanism body (PRO_MDO_BODY) for an advanced drag operation. This argument is relevant only for advanced drag types. For all other drag types, it is ignored and NULL should be passed.

The function ProKinDragReconnect() reconnects to a drag session taking into account constraints but not transforms from the active snapshot specified in ProKinDragEnvironmentSet(). Use ProWindowRefresh() to view the changes in the positions of the assembly components. Refer to the sections Snapshot Constraints on page 1180 and Snapshot Transforms on page 1182 for more information.

Performing Kinematic Drag

Function Introduced:

- **ProKinDragPerformMove()**

The function `ProKinDragPerformMove()` drags the selected geometric object to the specified X and Y screen coordinates given by `ProArray` of `Pro2dPnt` object. The geometric object can be `PRO_POINT`, `PRO_SURFACE_PNT`, `PRO_AXIS`, `PRO_EDGE`, `PRO_CURVE`, `PRO_DATUM_PLANE`, `PRO_SRF_PLANE_PNT`, or `PRO_SURFACE` in case of a point drag, and `PRO_PART` or `PRO_MDO_BODY` for all other drag types. A hook point is displayed on the object selected for drag, which is removed at the end of the drag operation. The dragging is performed according to the environment set by the function `ProKinDragEnvironmentSet()`. Use the constraints set for the active snapshot while dragging.

Creating and Modifying Snapshots

Functions Introduced:

- **ProKinDragSnapshotsNamesGet()**
- **ProSnapshotCreate()**
- **ProSnapshotUpdate()**
- **ProSnapshotRename()**
- **ProSnapshotDelete()**
- **ProSnapshotApply()**

The function `ProKinDragSnapshotsNamesGet()` retrieves an array of names of the snapshots belonging to the top-level assembly and all its subassemblies.

The function `ProSnapshotCreate()` creates a new snapshot for the top-level assembly as per its current position in the Creo Parametric window. Constraints are copied from the active snapshot. The newly created snapshot becomes active.

The function `ProSnapshotUpdate()` updates the snapshot for the top-level assembly as per its current position in the Creo Parametric window. Constraints are copied from the active snapshot. The newly updated snapshot becomes active.

The function `ProSnapshotRename()` renames the active snapshot of the top-level assembly.

The function `ProSnapshotDelete()` deletes a specified snapshot. The snapshot can be of the top-level assembly or any of its subassemblies. Pass `NULL` for the argument *path* to specify the top-level assembly.

The function ProSnapshotApply() applies the transforms of the active snapshot. Unlike in the user interface, no attempt to reconnect is made. Call ProKinDragReconnect() if reconnect is necessary. Use ProWindowRefresh() to view the changes in the positions of the assembly components.

Snapshot Constraints

The constraints that can be applied to a snapshot are contained by the structure ProSnapshotConstraint. The declaration for ProSnapshotConstraint is as follows:

```
typedef struct proSnapshotConstraint
{
ProSnapshotConstraintType   type;
ProSelection               *sel_array;
double                      value;
ProBool                     user_active;
ProBool                     valid
} ProSnapshotConstraint;
```

The fields in the above structure are described as follows:

- type—Specifies the type of snapshot constraint. The type is represented by the enumerated type ProSnapshotConstraintType and can take one of the following values:
  - PRO_SNAP_ALIGN—Select two points, two lines, or two planes from the top-level assembly. The two entities remain aligned during the drag operation.
  - PRO_SNAP_MATE—Select two planes from the top-level assembly. The planes remain mated during the drag operation.
  - PRO_SNAP_ORIENT—Select two planes that orient at an angle with each other.
  - PRO_SNAP_MOTION_AXIS_POS—Select a motion axis to specify the motion axis position.
  - PRO_SNAP_BODY_LOCK—Select the bodies to be locked together while dragging.
  - PRO_SNAP_CONNECTION_DISABLE—Select a connection that will be disabled while dragging.
  - PRO_SNAP_PARALLEL_VIEW_PLANE—Select a body that will move parallel only to the view plane. This constraint type is the same as the PRO_SNAP_ALIGN or PRO_SNAP_MATE types, but the second reference in this case is the view plane. This constraint type is available only via Creo Parametric TOOLKIT. If set by the Creo Parametric

TOOLKIT application, this constraint type becomes visible in the **Drag** dialog box in the Creo Parametric user interface, but it cannot be stored.

- PRO_SNAP_CAM_LIFTOFF_ENABLE—Allows two cams with a cam-follower connection to separate and collide during a dragging operation.
- PRO_SNAP_CAM_LIFTOFF_DISABLE—Requires two cams with a cam-follower connection to be in contact with each other during the dragging operation.

Note

The constraint types PRO_SNAP_CAM_LIFTOFF_ENABLE and PRO_SNAP_CAM_LIFTOFF_DISABLE override the Enable Liftoff property that you set from the **Cam-Follower Connection Definition** dialog box in the Creo Parametric user interface.

- *sel_array—Specifies the ProArray of selections. The number of selections needed and the permitted selection types for each constraint type are specified in the following table:

Constraint Type	Number of Selections needed	Permitted Selection Types
PRO_SNAP_ALIGN	2	PRO_POINT, PRO_SURFACE_PNT, PRO_AXIS, PRO_EDGE, PRO_CURVE, PRO_DATUM_PLANE, PRO_SRF_PLANE_PNT, PRO_SURFACE
PRO_SNAP_MATE, PRO_SNAP_ORIENT		PRO_DATUM_PLANE, PRO_SRF_PLANE_PNT, PRO_SURFACE
PRO_SNAP_MOTION_AXIS_POS	1	PRO_MDO_CONN_AXIS_ROT_1/2/3, PRO_MDO_CONN_AXIS_TR_1/2/3, PRO_MDO_SLOT_AXIS
PRO_SNAP_CONNECTION_DISABLE		PRO_MDO_CONN, PRO_MDO_CAM_CONN, PRO_MDO_SLOT_CONN, PRO_MDO_GEAR_CONN
PRO_SNAP_PARALLEL_VIEW_PLANE		PRO_POINT, PRO_SURFACE_PNT, PRO_EDGE_START, PRO_EDGE_END, PRO_CRV_START, PRO_CRV_END, PRO_AXIS, PRO_EDGE, PRO_CURVE, PRO_SRF_PLANE_PNT, PRO_SURFACE, PRO_DATUM_PLANE
PRO_SNAP_CAM_LIFTOFF_ENABLE		PRO_MDO_CAM_CONN
PRO_SNAP_CAM_LIFTOFF_DISABLE		
PRO_SNAP_BODY_LOCK	2 or more	PRO_PART, PRO_MDO_BODY

- value—Depending upon the constraint type, this field takes the following values:

- For the PRO_SNAP_ALIGN and PRO_SNAP_MATE constraint types, value specifies the linear distance between the references.
- For the PRO_SNAP_ORIENT type, value specifies the angle between the references.
- For the PRO_SNAP_MOTION_AXIS_POS type, value specifies the offset of the joint axis zero position. This value is angular for the rotation axis and linear for the translational axis. The references for the joint axis zero position may be default or as specified by the user.

- active—Specifies the ProBoolean option to enable or disable a constraint.
- valid—Specifies if the constraint is valid or invalid in the current model context. If this ProBoolean option is PRO_B_TRUE, the constraint is valid (active), and if it is PRO_B_FALSE, the constraint is invalid (suppressed).

Functions Introduced:

- **ProSnapshotConstraintsGet()**
- **ProSnapshotConstraintAdd()**
- **ProSnapshotConstraintDelete()**
- **ProSnapshotConstraintUpdate()**
- **ProSnapshotConstraintEvaluate()**

The function ProSnapshotConstraintsGet() retrieves all the constraints of a specified snapshot. The snapshot can be of the top-level assembly or any of its subassemblies. Pass NULL for the argument *path* to specify the top-level assembly.

The function ProSnapshotConstraintAdd() adds a constraint to the snapshot of the top-level assembly.

The function ProSnapshotConstraintDelete() deletes a constraint from the snapshot of the top-level assembly.

The function ProSnapshotConstraintUpdate() updates a constraint for the snapshot of the top-level assembly.

The function ProSnapshotConstraintEvaluate() calculates the position of the motion axis for the constraint type PRO_SNAP_MOTION_AXIS_POS for the active model in the Creo Parametric window. This value does not depend on the current snapshot and it is not necessary for the snapshot to contain the constraint.

Snapshot Transforms

Functions Introduced:

- **ProSnapshotTrfsGet()**
- **ProSnapshotTrfsUpdate()**

The function `ProSnapshotTrfsGet()` retrieves the transformation paths and transformation matrices saved in the snapshots for the subassemblies and their components with respect to the top-level assembly snapshot.

The function `ProSnapshotTrfsUpdate()` updates the transformation matrices saved in the snapshots for the subassemblies and their components with respect to the top-level assembly snapshot.

Snapshots in Drawing Views

Functions Introduced:

- **ProSnapshotAllowedInDrawingSet()**
- **ProSnapshotAllowedInDrawingGet()**

The function `ProSnapshotAllowedInDrawingSet()` assigns the active snapshot of the top-level assembly to be available or unavailable in drawings. Set the argument *allow* to `PRO_B_TRUE` to make the snapshot available in drawings.

The function `ProSnapshotAllowedInDrawingGet()` determines if the active snapshot is allowed in drawings. The argument *p_is_allowed* is `PRO_B_TRUE` if the snapshot is allowed in drawings.

57

Assembly: Simplified Representations

Creo Parametric TOOLKIT gives programmatic access to all the simplified representation functionality of Creo Parametric. You can create simplified representations either permanently or at runtime, and you can save, retrieve, or modify them by adding or deleting items.

Overview

Using Creo Parametric TOOLKIT, you can create and manipulate assembly simplified representations just as you can using Creo Parametric interactively.

Note

Creo Parametric TOOLKIT supports retrieval and activation of both part and assembly simplified representations. In addition, Creo Parametric TOOLKIT supports creation and modification of assembly simplified representations. Functions not appropriate for part mode are identified in the description.

Simplified representations are identified by the DHandle `ProSimprep`. As with other DHandles such as `ProFeature` and `ProGeomitem`, the `ProSimprep` handle contains just enough information to uniquely identify the object in the database—the model owner, type, and identifier.

The information required to create and modify a simplified representation is stored in a series of `ProSimprepdata` structures, which are visible data structures. The data structure contains the following fields:

- `ProName name`—The name of the simplified representation
- `ProBoolean temp`—Specifies whether it is a temporary, simplified representation
- `ProSimprepActionType action_type`—The rule that controls the default treatment of items in the simplified representation
- `ProSimprepitem *items`—An array of assembly components and features and the actions applied to them in the simplified representation

A `ProSimprepitem` is identified by the `ProIdTable` that defines the assembly component path to that item. (Even if the ID table path is only one level, use the `ProIdTable` and not the feature id for assemblies). Each `ProSimprepitem` has its own `ProSimprepAction` assigned to it. `ProSimprepAction` is a visible data structure that includes a variable of type `ProSimprepActionType`.

`ProSimprepActionType` is an enumerated type that specifies the possible treatment of items in a simplified representation. You can specify the following types of actions on the component:

- `PRO_SIMPREP_NONE`—Specifies that no action is specified.
- `PRO_SIMPREP_REVERSE`—Specifies that the reverse of the default rule must be applied to the component. For example consider that the default rule is to exclude a component. When you set the value `PRO_SIMPREP_REVERSE`, the component is included in the simplified representation.

- PRO_SIMPREP_INCLUDE—Specifies to include the component in the simplified representation.
- PRO_SIMPREP_EXCLUDE—Specifies to exclude the component in the simplified representation.
- PRO_SIMPREP_SUBSTITUTE—Specifies to substitute the component in the simplified representation.
- PRO_SIMPREP_GEOM—Specifies to use geometric representation.
- PRO_SIMPREP_GRAPHICS—Specifies to use graphical representation.
- PRO_SIMPREP_SYMB—Specifies to use symbolic representation.
- PRO_SIMPREP_BOUNDBOX—Specifies to use boundary box representation.
- PRO_SIMPREP_DEFENV—Specifies to use the default envelope representation.
- PRO_SIMPREP_LIGHT_GRAPH—Specifies to use light weight graphics representation.
- PRO_SIMPREP_AUTO—Specifies to use automatic representation.

Simplified Representations in Session

Functions Introduced:

- **ProSolidSimprepVisit()**
- **ProSimprepInit()**
- **ProSimprepSelect()**
- **ProSimprepActivate()**
- **ProSimprepActiveGet()**
- **ProSimprepTypeGet()**
- **ProSimprepIsDefault()**

This section describes the utility functions that relate to simplified representations.

ProSolidSimprepVisit() is like the other visit functions, and visits all the simplified representations of a parent ProSolid. The function visits only user-defined representation.

As all other visit functions, it takes four arguments—a pointer to the parent ProSolid, a filter function, the visit function itself, and a ProAppData field.

The function ProSimprepInit() initializes a ProSimprep structure. The function takes the following arguments:

- ProName*rep_name*— The name of the simplified representation in the solid. If you specify this argument, the function ignores the *rep_id*.
- int*rep_id*—The identifier of the simplified representation, if you did not specify rep_name (you specified NULL).
- ProSolid*p_solid*—The parent solid that contains the simplified representation.
- ProSimprep*p_simp_rep*—The handle to the newly initialized simplified representation.

The function ProSimprepSelect() creates a Creo Parametric menu to enable interactive selection. The function takes the parent solid as input, and outputs the handle to the selected simplified representation. If you choose the **Quit** menu button, the function returns the value PRO_TK_USER_ABORT. If the user selects the master representation, the returned simplified representation structure has an identifier of –1.

ProSimprepActivate() enables you to set the currently active simplified representation. To set a simplified representation to be the currently displayed model, you must also call ProSolidDisplay(). This function enables you to bring inactive submodels into memory, and use their handles without displaying them.

ProSimprepActivate() does not support activation of part simplified representations, because part simplified representation handles cannot be passed to this function. To obtain a handle to a part simplified representation use ProPartSimprepRetrieve(). You can display the simplified representation in a window using ProSolidDisplay().

ProSimprepActiveGet() enables you to find the currently active simplified representation. Given a model handle, ProSimprepActiveGet() returns the handle to the currently active simplified representation. If the current representation is the master representation, the identifier of the handle is set to –1.

The function ProSimprepTypeGet() returns the type of a specified simplified representation using the enumerated data type ProSimprepType:

- PRO_SIMPREP_MASTER_REP—Specifies a fully detailed assembly. The model tree lists all its components and identifies them as included, excluded, or substituted.
- PRO_SIMPREP_USER_DEFINED—Specifies a representation from the selected component.
- PRO_SIMPREP_GRAPH_REP—Specifies a representation that contains only information for display. You can quickly browse through a large assembly. Graphics representations cannot be modified or referenced.

- PRO_SIMPREP_GEOM_REP—Specifies a representations that contains complete component geometry information. As compared to graphics representations, geometry representations take longer to retrieve and require more memory.
- PRO_SIMPREP_SYMB_REP—Specifies a representation that allows you to represent components with a symbol.
- PRO_SIMPREP_DEFENV_REP—Specifies a representation that allows you to represent assembly components with an default envelope.
- PRO_SIMPREP_LIGHT_GRAPH_REP—Specifies a lightweight graphics representations of assemblies that contains assembly information and 3D thumbnail graphics representations of assembly components.
- PRO_SIMPREP_AUTO_REP—Specifies a representation for retrieving the minimum data that is required for presenting the assembly in the most accurate way. You can perform actions such as measuring distances between points on a light surface without retrieving the part geometry.

Note

When two standard representations of the same model are retrieved, for better memory usage, only one representation is used in the memory. The lower detailed representation is integrated into higher detailed representation. This higher detailed representation is used to retrieve both the representations.

If you retrieve a lower detailed representation when a higher detailed representation is already in the memory, this higher detailed representation is used and actually no retrieval is done.

The hierarchy for the representations is as follows with Master Simplified Representation being the highest representation level:

- Boundary Box Simplified Representation
- Symbolic Simplified Representation
- Graphic Simplified Representation
- Geometry Simplified Representation
- Master Simplified Representation

Refer to the Creo Parametric help for more information on Assembly Design.

The function ProSimprepIsDefault() determines if the specified simplified representation is the default representation for the owner model.

Retrieving Simplified Representations

Function Introduced:

- **ProAssemblySimprepMdlnameRetrieve()**
- **ProSimprepMdlnameRetrieve()**
- **ProMdlRepresentationFiletypeLoad()**

You can retrieve a named simplified representation from an assembly using the function `ProAssemblySimprepMdlnameRetrieve()`. This function retrieves the handle of an existing simplified representation from an assembly without getting the generic representation into memory.

The function takes as arguments—the names of the assembly and simplified representation, the representation data, the type of model to retrieve, and the handle to the assembly. Note that you must provide the name of the assembly. To retrieve an existing simplified representation, specify its name as one of the inputs to the argument of this function. The name of the simplified representation can be `NULL` if the representation data is provided. In this case, the instructions in the data are used to dynamically create a new simplified representation . The representation data can also be `NULL` if the name of the simplified representation is provided. Creo Parametric retrieves the simplified representation and any active submodels, and returns the `ProAssembly` handle.

You can retrieve geometry, graphics, symbolic simplified, boundary box, and default envelope representations into session using the function `ProSimprepMdlnameRetrieve()`. The input arguments to the function are:

- *model_name*—Specifies the name of the model whose simplified representation is to be retrieved.
- *file_type*—Specifies the type of model using the enumerated data type `ProMdlfileType`.
- *rep_type*—Specifies the type of simplified representation using the enumerated data type `ProSimprepType`.
- *rep_name*—Specifies the name of the simplified representation that must be retrieved.

.

Similar to `ProAssemblySimprepMdlnameRetrieve()`, the function `ProSimprepMdlnameRetrieve()` retrieves the simplified representation without bringing the master representation into memory. The function outputs the handle to the model. It does not display the simplified representation.

You can retrieve the simplified representation of a model into memory using the function `ProMdlRepresentationFiletypeLoad()`.

Retrieving and Expanding LightWeight Graphics Simplified Representations

Functions Introduced:

- **ProSimprepMdlnameRetrieve()**
- **ProLightweightGraphicsSimprepExpand()**

The function `ProSimprepMdlnameRetrieve()` retrieves the light graphics simplified representation of an assembly. Light graphics representations of assemblies contain assembly information and 3D thumbnail graphics representations of assembly components. In the light graphics simplified representation mode, the graphics of a model is represented using the Creo View viewable. The Creo View files must be in the same directory as the model. If the Creo View files are not available in the model directory, then the model is represented with bounding boxes in the light graphics simplified representation mode. Using this function you can initially retrieve and display graphic objects of higher levels of the assembly and then retrieve more detailed information of the sub-assemblies as required. This function is similar to `ProAssemblySimprepMdlnameRetrieve()`, and retrieves the light graphics assembly in the same way as described in the section Retrieving Simplified Representations on page 1189.

Use the function `ProLightweightGraphicsSimprepExpand()` to expand the light graphics representation of an assembly in the Creo Parametric active window to the specified level using the enumerated type `ProLightweightGraphicsSimprepLevel`.

You can expand the representation to the following levels:

- `PRO_LWG_SIMPREP_LEVEL_NEXT`—Expands thumbnails to the next level.
- `PRO_LWG_SIMPREP_LEVEL_ALL`—Expands thumbnails to all levels.

To expand the sub-assembly level, use the functions `ProModelitemInit()` and `ProSelectionAlloc()` to first initialize the `ProSelection` handle for the component part or sub-assembly. You can pass the `ProSelection` handle to the function `ProLightweightGraphicsSimprepExpand()` and expand the component part or sub-assembly node to the required level.

Retrieving User-Defined Simplified Representations

Functions Introduced:

- **ProAutomaticSimprepRetrieve()**
- **ProAutomaticSimprepConvert()**
- **ProAutomaticSimprepActivate()**

The function `ProAutomaticSimprepRetrieve()` retrieves a user-defined simplified representation as an automatic representation. If error occurs during regeneration, the assembly includes suppressed features. Use the function `ProSolidRetrievalErrorsGet()` to identify if any errors have occurred during retrieval of the simplified representation. The input arguments follow:

- *assem_name*—Name of the assembly specified using the structure `ProFamilyMdlName`.
- *file_type*—File type of the assembly specified using the enumerated data type `ProMdlfileType`.
- *simp_rep_name*—Name of the simplified representation.

The output argument `p_assem` is the handle to the assembly specified using the structure `ProAssembly`.

The function returns the error `PRO_TK_NO_PERMISSION` if the function does not have permission to operate on the specified assembly. The function returns the error `PRO_TK_E_NOT_FOUND` if the function did not find the specified simplified representation in the solid.

The function `ProAutomaticSimprepConvert()` converts a user-defined representation to automatic simplified representation while maintaining the excluded or substituted components in the representation.

The function `ProAutomaticSimprepActivate()` activates a user-defined representation as an automatic simplified representation. To display the correct simplified representation, you must also call the function `ProSolidDisplay()`.

Note

The functions `ProAutomaticSimprepRetrieve()`, `ProAutomaticSimprepConvert()` and `ProAutomaticSimprepActivate()` support only assemblies.

The function `ProAutomaticSimprepActivate()` returns the error `PRO_TK_E_NOT_FOUND` if the function did not find the specified simplified representation in the model.

Creating and Deleting Simplified Representations

Functions Introduced:

- **ProSimprepdataAlloc()**
- **ProSimprepCreate()**
- **ProSimprepDelete()**

> **Note**
>
> Creo Parametric TOOLKIT does not support creation of part simplified representations.

To create a simplified representation, you must allocate and fill a `ProSimprepdata` structure by calling the function `ProSimprepdataAlloc()`. As input, the function requires the name of the new simplified representation, the `temp` value, and the default rule. The specific structure is initialized by the function in the Creo Parametric database.

To generate the new simplified representation, call `ProSimprepCreate()`. This function returns the `ProSimprep` handle for the new representation.

The function `ProSimprepDelete()` deletes a simplified representation from its model owner. The function requires only the `ProSimprep` handle as input.

Example 1: Creating a Simplified Representation

The example in the file `UgSimprepCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_simprep`, shows how to create a simplified representation.

Extracting Information About Simplified Representations

Functions Introduced:

- **ProSimprepdataGet()**
- **ProSimprepdataFree()**
- **ProSimprepdataDefltGet()**
- **ProSimprepdataNameGet()**
- **ProSimprepdataTmpvalGet()**

- **ProSimprepdataitemsVisit()**
- **ProSimprepSubstitutionNameGet()**
- **ProAsmcompSubstitutionTypeGet()**
- **ProAsmcompSubstituteGet()**

> **Note**
>
> Creo Parametric TOOLKIT supports simplified representation of Assemblies only, not Parts.

Given the handle to a simplified representation and the address of a pointer to a `ProSimprepdata` structure, `ProSimprepdataGet()` fills out the `ProSimprepdata` structure. This function dynamically allocates storage for the data structure. When the memory is no longer needed, free it using the function `ProSimprepdataFree()`.

The `ProSimprepdataDefltGet()`, `ProSimprepdataNameGet()`, and `ProSimprepdataTmpvalGet()` functions return the associated values contained in the `ProSimprepdata` structure. They all take two arguments—the data structure to be queried, and the appropriate data structure for the type to be retrieved. `ProSimprepdataTmpvalGet()` retrieves the value of the `temp` field from the specified `ProSimprepdata`

The function `ProSimprepdataitemsVisit()` visits all the items that make up the simplified representation. The action and filter functions both have `ProSimprepitem*` as their first argument.

The function `ProSimprepSubstitutionNameGet()` returns the name of the substituted representation at the given assembly path even when the substituted representation is deleted from the model at the given path.

The function `ProAsmcompSubstitutionTypeGet()` returns the substitution type performed on the simplified representation of an assembly component. It takes the path to the component reprsentation that is being substituted, including the component ID, as one of its input arguments.

The function `ProAsmcompSubstituteGet()` returns the path to the substituted component representation in the form of and the handle to the substituted component representation in the form of `ProAsmcomp`. It takes the path to the component reprsentation that is being substituted, including the component ID, as one of its input arguments.

Example 2: Visiting the Items in a Simplified Representation

The sample code in `UgSimprepInfo.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_simprep` shows how to use the Creo Parametric TOOLKIT functions to visit the items in the specified simplified representation.

Modifying Simplified Representations

Functions Introduced:

- **ProSimprepActionInit()**
- **ProSimprepdataSet()**
- **ProSimprepdataDefltSet()**
- **ProSimprepdataNameSet()**
- **ProSimprepdataTmpvalSet()**

> **Note**
>
> Creo Parametric TOOLKIT supports simplified representation of Assemblies only, not Parts.

Using Creo Parametric TOOLKIT, you can modify the attributes of existing simplified representations. After you create or retrieve a simplified representation, you can make calls to the `ProSimprepdata*Set()` functions listed in this section to designate new values for the fields in the `ProSimprepdata` structure.

To modify an existing simplified representation, retrieve it, then get the handle to its `ProSimprepdata` structure by calling the function `ProSimprepdataGet()`. (If you created the representation programmatically within the same application, the `ProSimprepdata` handle is already available.) After modifying the data structure, reassign it to the corresponding simplified representation by calling the function `ProSimprepdataSet()`. Use the

function ProSimprepdataTmpvalSet to specify whether the newly created representation is temporary. Pass the value PRO_B_TRUE to the input argument temp to make the newly created representation a temporary representation.

Note

Use the function ProSimprepdataTmpvalSet() to set the value of the temp input argument in the specified ProSimprepdata structure while creating new simplified representations only. Once the simplified representation is created, this attribute is controlled by Creo Parametric. For all the existing representations, Creo Parametric controls the temp input argument and sets its value automatically

Adding Items to and Deleting Items from a Simplified Representation

Functions Introduced:

- **ProSimprepdataitemAdd()**
- **ProSimprepdataitemDelete()**
- **ProSimprepdataitemInit()**

Note

Creo Parametric TOOLKIT supports simplified representation of Assemblies only, not Parts.

You can add and delete items from the list of components in a simplified representation using Creo Parametric TOOLKIT. If you created a simplified representation using the option PRO_SIMPREP_EXCLUDE as the default rule, you would generate a list containing the items you want to include. Similarly, if the default rule for a simplified representation is PRO_SIMPREP_INCLUDE, you can add the items that you want to be excluded from the simplified representation to the list, setting the value of the ProSimprepActionType to PRO_SIMPREP_EXCLUDE.

To Add Items

1. Get the ProSimprepdata structure, as described in the previous section.
2. Specify the action to be applied to the item with a call to the function ProSimprepActionInit().

3. Initialize a `ProSimprepitem` structure for the item by calling the function `ProSimprepdataitemInit()`.
4. Add the item to the `ProSimprepdata` structure using the function `ProSimprepdataitemAdd()`.
5. Reassign the `ProSimprepdata` structure to the corresponding `ProSimprep` object by calling `ProSimprepdataSet()`.

To Remove Items

1. Get the `ProSimprepdata` structure handle.
2. Pass the `ProSimprepdata` handle and the `ProSimprepitem` handle for the item to be deleted to the function `ProSimprepdataitemDelete()`.
3. Reassign the `ProSimprepdata` structure to the corresponding `ProSimprep` object by calling the function `ProSimprepdataSet()`.

Gathering Components by Rule

Function Introduced:

- **ProRuleEval()**

Creo Parametric provides large assembly management tools. This section describes the access to some of this functionality through Creo Parametric TOOLKIT.

You can specify different types of rules and use them to generate a list of components for which the rule applies. After initializing the rule, call the function `ProRuleEval()` to generate the list of components that follow this rule.

Note that the returned list of components is in the form of an expandable array (`ProArray`), which is allocated by this function. To release the allocated memory, call the function `ProArrayFree()`.

The components can be gathered using the following rules:

- By model name
- By parameters, using an expression
- By location with a zone
- By distance from a point
- By size
- By an existing simplified representation

See the *Assembly Modeling User's Guide* for more details on this functionality.

Gathering by Model Name

Function Introduced:

- **ProRuleInitName()**

The function `ProRuleInitName()` initializes the rule for gathering by model name. The `name_mask` variable can be a wildcard. For more information, see the Creo Parametric help.

Gathering by Parameters

Function Introduced:

- **ProRuleInitExpr()**

You can specify a expression in the relations format to search for components of a particular parameter value. For example, consider the following expression:

```
type == "electrical" | cost <= 10
```

When you supply this expression to the rule, it gathers the components that have a "cost" parameter of less than or equal to 10, or whose *type* parameter is set to "electrical."

The*expr* variable is an array of `ProLine` structures. You allocate this array using the function `ProArrayAlloc()`. The `ProArray*` functions are used for all array manipulations.

Gathering by Zone

Function Introduced:

- **ProRuleInitZone()**

When you specify this rule, all the components that belong to the supplied zone feature are gathered.

See the *Assembly Modeling User's Guide* for detailed information about setting up and working with zones.

When you create a zone, the function creates a feature of type `PRO_FEAT_ZONE` in the top-level assembly.

Gathering by Distance from a Point

Function Introduced:

- **ProRuleInitDist()**

Using `ProRuleInitDist()` to set up a rule that specifies distance from a point, Creo Parametric TOOLKIT gathers all the components within the specified spherical region.

By filling the ProRuleDist data structure, you can specify the center and the distance from the center. This information is in the coordinates of the top-level assembly.

Gathering by Size

Function Introduced:

- **ProRuleInitSize()**

By filling the ProRuleSize data structure, you can specify the size of components to be gathered.

If you want to gather the components greater than the specified size, set the field greater to PRO_B_TRUE. If you set the field to PRO_B_FALSE, the function gathers the components that are less than the specified size.

If you want the specified size to be in absolute terms, set the field absolute to PRO_B_TRUE. Note that in this case, the function uses the units of the top-level assembly.

If the information is relative, set the field absolute to PRO_B_FALSE. In this case, the only valid values that can be specified are in the range (0.0, 1.0). The function compares the component size to that of the top-level assembly, and uses this ratio to determine whether the component should be gathered.

Gathering by Simplified Representation

Function Introduced:

- **ProRuleInitRep()**

You can gather components that belong to an existing simplified representation by calling the function ProRuleInitRep(), which initializes the rule.

58

Assembly: Data Sharing Features

This chapter describes the Creo Parametric TOOLKIT functions access for Data Sharing Features (DSF). These types of features are used to transfer information (geometry, annotations, and other details) from one model to another. Data Sharing Features also help you to consolidate your design information in a central location, control change propagation which aid towards accomplishing top down design objectives. Data Sharing Features are of the following types:

- Copy Geometry / Publish Geometry
- Shrinkwrap
- Merge / Cutout
- Inheritance

It also explains how to access the properties of variant features and lists read and write functions supporting the Inheritance Features.

Copy Geometry, Publish Geometry, and Shrinkwrap Features

Copy Geometry Features are used to pass any type of geometric reference information, user-defined parameters to and from parts, skeleton models, and assemblies. Copy Geometry features can only copy reference geometry such as surface, datum features and not solid geometry. They are used in a top-down design to reduce the amount of data in session, thus avoiding the retrieval of entire reference source models.

A Publish Geometry feature contains independent, local geometry references. Only local geometry can be referenced in a Publish Geometry feature and external references are not allowed. A Publish Geometry feature has no geometry and does not create local copies of the references selected for its definition. It simply consolidates multiple local references in a model so that they can be copied to other models.

A Shrinkwrap feature is a collection of surfaces and datum features of a model that represents the exterior of the model. You can use a part, skeleton, or top-level assembly as the source model for a Shrinkwrap feature. A Shrinkwrap feature is associative and automatically updates to reflect changes in the parent copied surfaces.

Feature Element Tree for the Copy Geometry, Publish Geometry, and Shrinkwrap Features

The element tree for Copy Geometry, Publish Geometry, and Shrinkwrap features is documented in the header file `ProDataShareFeat.h`.

The following figure demonstrates the feature element tree structure:

Feature Element Tree for Copy Geometry, Publish Geometry, and Shrinkwrap Features

```
PRO_E_FEATURE_TREE
|
|-PRO_E_FEATURE_TYPE
|-PRO_E_STD_FEATURE_NAME
|-PRO_E_CG_FEAT_SUB_TYPE
|-PRO_E_CG_REFS_TYPE
|-PRO_E_CG_LOCATION
|      |-PRO_E_DSF_EXT_LOCAL_TYPE
|      |-PRO_E_DSF_SEL_REF_MDL
|      |-PRO_E_CG_PLACEMENT
|           |-PRO_E_CG_PLACE_TYPE
|           |-PRO_E_CG_CSYS_PLACE
|           |     |-PRO_E_CG_PLC_LOCAL_CSYS
|           |     |-PRO_E_CG_PLC_EXT_CSYS
|           |-PRO_E_CG_FOLLOW_SRF_OPT
|-PRO_E_CG_PG_OR_REFS
|     |-PRO_E_CG_PUBD_GEOM
|     |-PRO_E_CG_REFS_COLL
|          |-PRO_E_STD_SURF_COLLECTION_APPL
|          |-PRO_E_STD_CURVE_COLLECTION_APPL
|     |-PRO_E_CG_OBJS_COLL
|     |-PRO_E_CG_BODY_COLL
|-PRO_E_SW_COLLECTION_TYPE
|-PRO_E_SW_OPTIONS
|    |-PRO_E_SW_QUALITY
|    |-PRO_E_SW_FILL_HOLES
|    |-PRO_E_SW_COLLECT_QUILTS
|    |-PRO_E_SW_SKIP_SURF_SIZE
|    |-PRO_E_SW_COLLECT_ORDER
|    |-PRO_E_SW_RES_GEOM_OPT
|    |-PRO_E_SW__FAILED_SLD_OPT
|    |-PRO_E_SW_FILL_CNTRS_ARR
|    |   |-PRO_E_SW_FILL_CNTRS
|    |      |-PRO_E_SW_FILL_CNTRS_SRF_SEL
|    |-PRO_E_SW_FILL_CNTRS_DISP_CRV
|-PRO_E_SW_COMP_SUBSET
|    |-PRO_E_SW_COMPONENT
|        |-PRO_E_SW_INCLUDE_COMP
|        |-PRO_E_SW_SEL_COMPONENT
|-PRO_E_SW_REFS_COLL
|      |-PRO_E_STD_SURF_COLLECTION_APPL
|      |-PRO_E_SW_EXCLUDE_SURF_COLL_APPL
|      |-PRO_E_STD_CURVE_COLLECTION_APPL
|      |-PRO_E_CG_OBJS_COLL
|-PRO_E_DSF_PROPAGATE_ANNOTS
|-PRO_E_CG_SRFS_COPY
|     |-PRO_E_SRF_COPY_TYPE
|     |-PRO_E_SRF_COPY_EXCL
|     |-PRO_E_SRF_COPY_FILL
|     |-PRO_E_STD_CURVE_COLLECTION_APPL
|--PRO_E_DSF_DTMS_FIT
|-PRO_E_DSF_DEPENDENCY
|-PRO_E_DSF_NOTIFY_UPDATE
```

PRO_E_CG_LOCATION

```
|-PRO_E_CG_LOCATION
|      |-PRO_E_DSF_EXT_LOCAL_TYPE
|      |-PRO_E_DSF_SEL_REF_MDL
|      |-PRO_E_CG_PLACEMENT
|           |-PRO_E_CG_PLACE_TYPE
|           |-PRO_E_CG_CSYS_PLACE
|           |     |-PRO_E_CG_PLC_LOCAL_CSYS
|           |     |-PRO_E_CG_PLC_EXT_CSYS
|           |-PRO_E_CG_FOLLOW_SRF_OPT
```

PRO_E_DSF_PROPAGATE_ANNOTS

```
PRO_E_DSF_PROPAGATE_ANNOTS
    |-PRO_E_DSF_ANNOT_CPY_ALL
    |-PRO_E_DSF_ANNOT_DEPEND_ALL
    |-PRO_E_DSF_ANNOT_AUTO_CPY_DTM
    |-PRO_E_DSF_ANNOT_SELECTIONS
         |-PRO_E_DSF_ANNOT_SELECTION
              |-PRO_E_DSF_ANNOT_SEL_ANNOTS
              |-PRO_E_DSF_ANNOT_CPY_STATUS
              |-PRO_E_DSF_ANNOT_DEPENDENCY
```

PRO_E_DSF_DTM_FIT

```
PRO_E_DSF_DTM_FIT
     |-PRO_E_DSF_DTM_SELECTION
     |-PRO_E_DTMPLN_FIT
     |     |-PRO_E_DTMPLN_FIT_TYPE
     |     |-PRO_E_DTMPLN_FIT_REF
     |     |-PRO_E_DTMPLN_FIT_DTM_RAD
     |     |-PRO_E_DTMPLN_FIT_OUTLINE
     |-PRO_E_DTMAXIS_FIT
          |-PRO_E_DTMAXIS_FIT_TYPE
          |-PRO_E_DTMAXIS_FIT_REF
          |-PRO_E_DTMAXIS_FIT_LEN
```

The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_GEOM_COPY`, for Copy Geometry, Publish Geometry, and Shrinkwrap features.
- `PRO_E_CG_FEAT_SUB_TYPE`—Specifies the sub feature type and is a mandatory element. It is visible for all copy geometry features. The valid sub types are as follows:

- PRO_CG_COPY_GEOM—Copy Geometry feature
- PRO_CG_SHRINKWRAP—Shrinkwrap feature
- PRO_CG_PUB_GEOM—Publish Geometry feature

- PRO_E_CG_REFS_TYPE—Specifies the type of references used in a copy geometry feature (a published geometry feature, or an selected array of surfaces, edges, curves and datums). It is visible for Copy Geometry features.
- PRO_E_CG_LOCATION—Specifies a compound element that indicates the method used for placement of the feature within the parent model. This is valid for Copy Geometry and Shrinkwrap features. See the section describing Element Details of the Subtree PRO_E_CG_LOCATION on page 1207 for more information.
- PRO_E_CG_PG_OR_REFS—Specifies either a published geometry feature to copy or a collection of local geometry references to copy. It is visible for Copy Geometry features and has the following elements:
  - PRO_E_CG_PUBD_GEOM—Specifies the selected publish geometry feature to be copied.
  - PRO_E_CG_REFS_COLL—Specifies the collection of references. It is used for Copy Geometry and Publish Geometry features and has the following elements:
    - PRO_E_STD_SURF_COLLECTION_APPL—Specifies a collection of selected surfaces to copy.
    - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies a collection of selected curves and / or edges to copy.
  - PRO_E_CG_OBJS_COLL—Specifies a multivalued element containing miscellaneous references such as datums, quilts, points etc.
  - PRO_E_CG_BODY_COLL—Specifies a multivalued element that contains body selections to copy.

Note

The reference collection elements PRO_E_STD_SURF_COLLECTION_APPL, PRO_E_STD_CURVE_COLLECTION_APPL, PRO_E_CG_OBJS_COLL may return the item of type PRO_QUERY. If PRO_QUERY is encountered, the Creo Parametric TOOLKIT application cannot access or modify the rules of the query. However, the application can redefine the other properties of the DSF without affecting or removing the query. For more information refer to the section Saved Queries for Copy Geometry and Publish Geometry Features on page 1210.

- PRO_E_SW_COLLECTION_TYPE—Specifies the collection mode used while creating a Shrinkwrap feature. The values for this element, specified by the enumerated type ProShrinkwrapCollectionType, are as follows:
  - PRO_SW_OUTER_SHELL—Surfaces and datums that represent the exterior of the model are used. This is the default.
  - PRO_SW_ALL_SOLID_SURFS—All solid surfaces in the model are automatically collected.
  - PRO_SW_MANUAL—Surfaces, edges, curves, and datums that you select are used. You can copy geometry from more than one reference model.
- PRO_E_SW_OPTIONS—Specifies Shrinkwrap feature options. This element is visible for Shrinkwrap features and has the following elements:
  - PRO_E_SW_QUALITY—Specifies the shrinkwrap quality level used when identifying the contributing geometry to the Shrinkwrap feature. It can have valid values in the range of 1 to PRO_MAX_SHRINKWRAP_QUALITY_LVL.
  - PRO_E_SW_FILL_HOLES—Specifies whether or not to use the option to auto fill holes.
  - PRO_E_SW_COLLECT_QUILTS—Specifies whether or not to include external quilts in the Shrinkwrap feature.
  - PRO_E_SW_SKIP_SURF_SIZE—Specifies the shrinkwrap skip surfaces size. Creo Parametric will not include surfaces smaller than the specified percentage of the model's size in the Shrinkwrap model. It can have a value ranging from 0(default) to 100%, to specify the relative size of the surface to ignore.
  - PRO_E_SW_COLLECT_ORDER—Specifies how the system will handle the subcomponents in creating the feature. This value can be of the following types:
    - PRO_SW_SHRINKWRAP_AND_SELECT—Specifies to first shrinkwrap and then select (selected by default). The system analyzes the entire assembly to identify the external surfaces to be included and only the appropriate surfaces that belong to the selected components are included in the resulting shrinkwrap feature.
    - PRO_SW_SELECT_AND_SHRINKWRAP—Specifies select and shrinkwrap. The system builds a shrinkwrap based on selected components.
  - PRO_E_SW_RES_GEOM_OPT—Specifies the options for the resultant Shrinkwrap geometry. This element is available only if the element PRO_E_SW_COLLECTION_TYPE is set to the value PRO_SW_ALL_SOLID_

SURFS. The values for this element, specified by the enumerated type ProShrinkwrapResGeomOpt, are as follows:

- PRO_SW_RES_GEOM_QUILT—The resulting Shrinkwrap feature is a quilt.
- PRO_SW_RES_GEOM_SOLID—The resulting Shrinkwrap feature is a solid.
- PRO_SW_RES_GEOM_ASM_QUILT—The resulting Shrinkwrap feature is a quilt that contains a merged geometry of a referenced assembly.

  - PRO_E_SW__FAILED_SLD_OPT—Specifies the options to address the failed solidification subfeatures (external reference copy geometry features) of a Shrinkwrap feature. This element is available only if the element PRO_E_SW_RES_GEOM_OPT is set to the value PRO_SW_RES_GEOM_SOLID. The values for this element, specified by the enumerated type ProShrinkwrapFailedSldOpt, are as follows:
    - PRO_SW_FAILED_SLD_FAIL—Failed solidification subfeatures of the Shrinkwrap feature are not resolved. As a result, the Shrinkwrap feature also fails.
    - PRO_SW_FAILED_SLD_TO_QUILT—Failed solidification subfeatures of the Shrinkwrap feature are restored as quilts. As a result, the Shrinkwrap feature does not fail.
  - PRO_E_SW_FILL_CNTRS_ARR—Specifies the contours of open spaces that need to be filled. This element consists of an array of compound elements of the type PRO_E_SW_FILL_CNTRS which consists of the following elements:
    - PRO_E_SW_FILL_CNTRS_SRF_SEL—Select the surface that defines the contour to be filled.
    - PRO_E_SW_FILL_CNTRS_DISP_CRV—Displays a yellow curve instead of a filled contour.

- PRO_E_SW_COMP_SUBSET—Specifies components of the assembly to be considered when creating the Shrinkwrap. This array element consists of an array of Shrinkwrap component subset elements (PRO_E_SW_COMPONENT). That element includes the following elements:
  - PRO_E_SW_INCLUDE_COMP specifies whether or not to include the component in the shrinkwrap
  - PRO_E_SW_SEL_COMPONENT specifies the selected component.

- PRO_E_SW_REFS_COLL—Specifies the collection of surfaces and other references to be included or excluded from the Shrinkwrap feature. It has the following elements:
  - PRO_E_STD_SURF_COLLECTION_APPL—Specifies the collection of individual surfaces that must always be included in the Shrinkwrap feature.
  - PRO_E_SW_EXCLUDE_SURF_COLL_APPL—Specifies the collection of individual surfaces that must always be excluded from the Shrinkwrap feature.
  - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies a collection of chains constructed by selection of edges or curves.
  - PRO_E_CG_OBJS_COLL—Specifies a multivalued element containing miscellaneous references such as datums, quilts, points etc.
- PRO_E_DSF_PROPAGATE_ANNOTS—Specifies rules about how to propagate annotations. See the section Element Details of PRO_E_DSF_PROPAGATE_ANNOTS on page 1208 below for the structure and contents of this element.
- PRO_E_CG_SRFS_COPY—Specifies a compound element that specifies copied surfaces. It is visible for Copy Geometry features and has the following elements:
  - PRO_E_SRF_COPY_TYPE—Specifies the type of copied surface. It can have one of the following values:
    - PRO_SRFCOPY_AS_IS
    - PRO_SRFCOPY_EXCLD_FILL
    - PRO_SRFCOPY_INSIDE_BNDRY
    - PRO_SRFCOPY_UNTRIM_TO_ENVLP
    - PRO_SRFCOPY_UNTRIM_TO_DOMAIN
  - PRO_E_SRF_COPY_EXCL—Specifies excluded surfaces.
  - PRO_E_SRF_COPY_FILL—Specifies loops to fill.
  - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the boundary for the copied surfaces.
- PRO_E_DSF_DTMS_FITS—Specifies a compound element that indicates the rules for fitting datums in the DSF feature. It is visible for all internal Copy Geometry features) and Shrinkwrap features. This compound element includes PRO_E_DTMPLN_FIT and PRO_E_DTMAXIS_FIT subtrees for any or all of the copied datums. Refer to the chapter Element Trees: Datum Features on page 804 for details on the datum fit subtrees.

- PRO_E_DSF_DEPENDENCY—Specifies the dependency type of the Copy Geometry Feature. The values for this element are specified by the enumerated type ProDSFDependency defined in the header file ProDataShareFeat.h. For more information on the values, refer to the section Feature Element Tree on page 1211.

  The valid values for the dependency status are as follows:

  - PRO_DSF_UPDATE_AUTOMATICALLY—Specifies that the geometry can be updated when its parent changes.
  - PRO_DSF_UPDATE_MANUALLY—Suspends the relationship between the current feature and the original geometry. If you change the original part, the current feature does not update automatically. It has to be updated manually.
  - PRO_DSF_NO_DEPENDENCY—There is no dependency between DSF feature and referenced model.

  Note

  From Creo Parametric 3.0 onward, the enumerated type ProDsfDependency has been deprecated. Use the enumerated type ProDSFDependency instead.

- PRO_E_DSF_NOTIFY_UPDATE—Specify the notification status for the specified feature using the enumerated value ProDsfNotifyUpdate. Use this element only if the element PRO_E_DSF_DEPENDENCY is set to the value PRO_DSF_UPDATE_MANUALLY. The valid values for this element are:

  - PRO_DSF_NOTIFY_UPDATE_OFF—Switches off the notification update. This is the default value
  - PRO_DSF_NOTIFY_UPDATE_ON—Switches on the notification update.

  Use the functions ProFeatureDSFDependencyNotifySet() and ProFeatureDSFDependencyNotifyGet() to set and get the notification status of a DSF feature. For more information on these functions, refer to the section Feature and CopyGeom Feature Functions on page 1153 in the chapter Assembly: Top-down Design.

Element Details of the Subtree PRO_E_CG_LOCATION

The compound element PRO_E_CG_LOCATION has the following elements:

- PRO_E_DSF_EXT_LOCAL_TYPE—Specifies the DSF location type:

- PRO_DSF_PLACE_LOCAL type is a local feature within the assembly. Therefore the references will be dependent upon the assembly structure and the feature does not need placement information.
- PRO_DSF_PLACE_EXTERNAL type is set to externalize a Data Sharing Feature. An external data-sharing feature must be placed in its target model explicitly.

Note

The purpose of External DSF's is to copy geometry from one model to another model without the need to copy the geometry in the context of the assembly. External DSF's reduce the dependency on the assembly and all models along the path between the two components. Once a feature has been made "External", it cannot be converted to become internal.

- PRO_E_DSF_SEL_REF_MDL—Specifies the model to use for the external DSF.
- PRO_E_CG_PLACEMENT—Specifies the placement of the external reference model in the target model. It has the following elements:
  - PRO_E_CG_PLACE_TYPE—The external placement reference for the copied geometry can be of the following types:
    - PRO_CG_PLC_DEFAULT—Locates the copied geometry in the current model using the default location.
    - PRO_CG_PLC_CSYS_CSYS—Locates the copied geometry in the current model by aligning coordinate systems.
    - PRO_CG_PLC_CURRENT—Locates the copied geometry in the current using the current placement (applicable only during a conversion of a local DSF to an external DSF).
    - PRO_E_CG_CSYS_PLACE—The two reference elements below this element specifies Csys-Csys alignment.
- PRO_E_CG_FOLLOW_SRF_OPT—Specifies the options for surface to be followed.

Element Details of PRO_E_DSF_PROPAGATE_ANNOTS

The compound element describes the options available to propagate annotations in DSF's. It has the following subelements:

- PRO_E_DSF_ANNOT_CPY_ALL—Specifies a flag whether to copy all annotation elements.
- PRO_E_DSF_ANNOT_DEPEND_ALL—Specifies a flag whether to make the copied annotation elements as dependent on their originals.
- PRO_E_DSF_ANNOT_AUTO_CPY_DTM—Specifies a flag to propagate annotation planes and other datums referenced by annotation elements automatically. Annotation planes are propagated if the annotation reference comprising of the solid or surface geometry is copied.
- PRO_E_DSF_ANNOT_SELECTIONS—Instead of using the automatic flags for propagation, the DSF feature can specify a list of annotations to propagate. Each subelement representing an annotation has the following sub-elements:
  - PRO_E_DSF_ANNOT_SEL_ANNOTS specifies the manually selected annotation element
  - PRO_E_DSF_ANNOT_CPY_STATUS specifies the copy status for this selected element.
  - PRO_E_DSF_ANNOT_DEPENDENCY specifies whether to make the annotation dependant in the DSF. It can have one of the following values:
    - PRO_DSF_DEPENDENT is set by default.
    - PRO_DSF_INDEPENDENT makes the annotations independent of changes made to the parent annotation.

Shrinkwrap Features Created from Copy Geometry References

Functions Introduced:

- **ProFeatureIsShrinkwrapRefCopyGeom()**
- **ProFeatureShrinkwrapGetRefCopyGeoms()**
- **ProFeatureIsShrinkwrap()**
- **ProFeatureRefCopyGeomShrinkwrapGet()**

The function ProFeatureIsShrinkwrapRefCopyGeom() checks whether the specified feature is a reference copy geometry feature, which is created by the shrinkwrap feature. The shrinkwrap reference copy geometry features reference models, which are collected by shrinkwrap according to the type of shrinkwrap, user-defined options, and user selections.

Use the function ProFeatureShrinkwrapGetRefCopyGeoms() to get an array of reference copy geometry features, which are created by the shrinkwrap feature.

The function `ProFeatureIsShrinkwrap()` checks whether the specified feature is a shrinkwrap feature.

Use the function `ProFeatureRefCopyGeomShrinkwrapGet()` to get the shrinkwrap feature for the specified reference copy geometry feature.

Saved Queries for Copy Geometry and Publish Geometry Features

Copy Geometry and Publish Geometry features have the ability to retain and reuse search tool queries defined from its collectors. The functions described in this section provide the ability to update these query-driven data sharing features using Creo Parametric TOOLKIT.

Functions Introduced:

- **ProDatasharingfeatureIsQuerydriven()**
- **ProDatasharingfeatureQueryUpdate()**

The function `ProDatasharingfeatureIsQuerydriven()` returns true if the specified data sharing feature is query driven.

The function `ProDatasharingfeatureQueryUpdate()` updates the items collected by the query within the Copy Geometry or Publish Geometry feature. This will regenerate the feature and may cause geometry to be added or removed.

Retrieving a copy of the annotation item

Functions Introduced:

- **ProDatasharingfeatCopiedAnnotFind()**

The function `ProDatasharingfeatCopiedAnnotFind()` retrieves the annotation item owned by the data sharing feature, which is a copy of specified annotation item. The function supports Inheritance, Merge, CopyGeom, ShrinkWrap, and Cutout features only. The input arguments are:

- *p_datasharing_feature*—Specifies the data sharing feature.
- *p_orig_path*—Specifies the component path from the top level assembly to the subcomponent that owns the annotation. Pass `NULL` if the annotation is owned by the top level model in the data sharing feature.
- *p_orig_item*—Specifies the annotation item in the original model.

General Merge (Merge, Cutout and Inheritance Feature)

Feature Element Tree

The element tree for the general merge feature is documented in the header file `ProDataShareFeat.h`. The following figure demonstrates the feature element tree structure:

Feature Element tree for General Merge Feature

```
PRO_E_FEATURE_TREE
|
|-PRO_E_FEATURE_TYPE
|-PRO_E_STD_FEATURE_NAME
|-PRO_E_GMRG_FEAT_TYPE
|-PRO_E_DSF_REF_MDL
|    |--PRO_E_DSF_EXT_LOCAL_TYPE
|    |--PRO_E_DSF_SEL_REF_MDL
|    |--PRO_E_DSF_PLACEMENT
|-PRO_E_GMRG_MATERIAL_OPT
|-PRO_E_GMRG_VARIED_ITEMS
|-PRO_E_GMRG_COPY_DATUMS
|-PRO_E_DSF_PROPAGATE_ANNOTS
|--PRO_E_DSF_DTMS_FIT
|       |-PRO_E_DSF_DTM_FIT
|-PRO_E_DSF_DEPENDENCY
|-PRO_E_DSF_NOTIFY_UPDATE
|-PRO_E_IS_SMT_CUT
|-PRO_E_SMT_CUT_NORMAL_DIR
```

PRO_E_DSF_PLACEMENT

```
PRO_E_DSF_PLACEMENT
   |--PRO_E_COMPONENT_CONSTRAINTS
       |--PRO_E_COMPONENT_CONSTRAINT
           |--PRO_E_COMPONENT_CONSTR_TYPE
           |--PRO_E_COMPONENT_COMP_CONSTR_REF
           |--PRO_E_COMPONENT_ASSEM_CONSTR_REF
           |--PRO_E_COMPONENT_CONSTR_REF_OFFSET
           |--PRO_E_COMPONENT_USER_DATA
           |--PRO_E_COMPONENT_CONSTR_ATTR
           |--PRO_E_COMPONENT_COMP_ORIENT
           |--PRO_E_COMPONENT_ASSM_ORIENT
           |--PRO_E_COMPONENT_CONSTR_SET_ID
           |--PRO_E_COMPONENT_SLOT_EXTRA_CRV_REF
```

The following list details special information about some of the elements in this tree:

- PRO_E_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_GEN_MERGE.
- PRO_E_GMRG_FEAT_TYPE—Specifies the type of General Merge Feature:
  - PRO_GEN_MERGE_TYPE_MERGE (Merge or cutout feature)
  - PRO_GEN_MERGE_TYPE_INHERITANCE (Inheritance feature)
- PRO_E_DSF_REF_MDL—Specifies the reference model. It has the following elements:
  - PRO_E_DSF_EXT_LOCAL_TYPE—Specifies the DSF location type and is of the following type:
    - PRO_DSF_PLACE_LOCAL type is a local reference to the reference model
    - PRO_DSF_PLACE_EXTERNAL indicates an external merge, cutout, or inheritance feature.
    - PRO_E_DSF_SEL_REF_MDL—Specifies a selected reference model.
    - PRO_E_DSF_PLACEMENT—Specifies the placement of the Data Sharing feature. It contains the following elements:
      ? PRO_E_COMPONENT_CONSTRAINTS

For more information on component constraint elements, refer to chapter Assembly: Assembling Components on page 1159.

- PRO_E_GMRG_MATERIAL_OPT—Specifies the general material options. These are listed as follows:
  - PRO_GEN_MERGE_RMV_MATERIAL—It removes material from the modified (target) model . The material removed is equal to the modifying (source) model.
  - PRO_GEN_MERGE_ADD_MATERIAL—It adds material to the modified model from the modifying model. The material added is equal to the modifying model.
  - PRO_GEN_MERGE_INT_MATERIAL—It retains the intersecting material between the modified and modifying models.

  Refer to Creo Parametric Assembly Help for more information.
- PRO_E_GMRG_VARIED_ITEMS—Specifies a pointer element that defines the inheritance feature varied items and their values. This handle cannot be directly read or modified by Creo Parametric TOOLKIT. Instead, use the Visit functions available in ProVariantFeat.h to read varied items. To set varied items (after the feature has been created), use the appropriate modification function on an item whose owner is the variant feature model handle (ProVariantfeatMdlGet()). For more information, refer to the section Inheritance Feature and Flexible Component Variant Items on page 1215.
- PRO_E_GMRG_COPY_DATUMS—True to copy datums with this merge or inheritance feature, false to leave them uncopied.
- PRO_E_DSF_PROPAGATE_ANNOTS—Specifies rules about how to propagate annotations. Refer to the section Feature Element Tree for the Copy Geometry, Publish Geometry, and Shrinkwrap Features on page 1200 for the details. A Shrinkwrap feature is a collection of surfaces and datum features of a model that represents the exterior of the model. You can use a part, skeleton, or top-level assembly as the source model for a Shrinkwrap feature. A Shrinkwrap feature is associative and automatically updates to reflect changes in the parent copied surfaces.
- PRO_E_DSF_DTMS_FIT—Specifies a compound element that indicates the rules for fitting datums in the DSF feature. It is visible for all internal Copy Geometry features) and Shrinkwrap features. This compound element includes PRO_E_DTMPLN_FIT and PRO_E_DTMAXIS_FIT subtrees for any or all

of the copied datums. Refer to the chapter Element Trees: Datum Features on page 804 for details on the datum fit subtrees.

- `PRO_E_DSF_DEPENDENCY`—Specifies the dependency type. The values for this element are specified by the enumerated type `ProDsfDependency`.

> **Note**
>
> From Creo Parametric 3.0 onward, the enumerated type `ProDsfDependency` has been deprecated. Use the enumerated type `ProDSFDependency` instead.

The types of dependencies are:

- `PRO_DSF_UPDATE_AUTOMATICALLY`—Specifies that the geometry of the DSF feature depends upon the geometry of the parent model used during feature creation. The DSF feature reflects all the changes made in the parent model.

> **Note**
>
> From Creo Parametric 3.0 onward, the value `PRO_DSF_DEPENDENT` has been deprecated. Use the enumerated value `PRO_DSF_UPDATE_AUTOMATICALLY` instead.

- `PRO_DSF_UPDATE_MANUALLY` —Specifies that the geometry of the DSF feature is independent of the geometry of the parent model used during feature creation. If you update the parent model, the DSF feature does not change.

> **Note**
>
> From Creo Parametric 3.0 onward, the value `PRO_DSF_INDEPENDENT` has been deprecated. Use the enumerated value `PRO_DSF_UPDATE_MANUALLY` instead.

- `PRO_DSF_NO_DEPENDENCY`—Specifies that there is no dependency between the geometry of the DSF feature and the geometry of the parent model used during feature creation.

PRO_E_DSF_NOTIFY_UPDATE—Specifies the notify status in the specified feature using the enumerated value ProDsfNotifyUpdate. For more information on this element, refer to the section Feature Element Tree for the Copy Geometry, Publish Geometry, and Shrinkwrap Features on page .

PRO_E_IS_SMT_CUT—Specifies whether the specified feature is a sheetmetal cut or a solid cut. If true this feature is a sheetmetal cut.

PRO_E_SMT_CUT_NORMAL_DIR—Specifies the surface to which the section projection will be normal.

Note

For more information,, refer to the section Sheetmetal Cut Features on page 1345 in the chapter Production Applications: Sheetmetal.

Inheritance Feature and Flexible Component Variant Items

An Inheritance feature allows one-way associative propagation of geometry and feature data from a reference part to target part within an assembly. The reference part is the original part and the target part contains the inheritance features. Inheritance features are always created by referencing existing parts. An inheritance feature begins with all of its geometry and data identical to the reference part from which it is derived.

Users can vary the visibility and values of items in inheritance features. Creo Parametric TOOLKIT offers the ability to access these varied items and their properties using the regular Creo Parametric TOOLKIT function appropriate for the property. Creo Parametric TOOLKIT also provides the ability to locate and separate the varied properties from the non-varied ones.

A flexible component has similar capabilities for variance of dimensions and parameters of a model in the context of an assembly. Creo Parametric TOOLKIT supports access to the variant properties of flexible components through the regular Creo Parametric TOOLKIT functions as well.

This section refers collectively to inheritance features and flexible components as "variant features".

Variant Feature Model

Functions Introduced:

- **ProVariantfeatMdlGet()**
- **ProMdlIsVariantfeatMdl()**
- **ProMdlVariantfeatAsmcomppathGet()**

Use the function `ProVariantfeatMdlGet()` to obtain the special model pointer from a variant feature. A special `ProMdl` handle represents the "inherited" or "flexible" model handle in functions which access variant features. This handle is called the "Variant Feature Model". Only certain functions will support inputs including the Variant Feature Model handle.

The function `ProMdlVariantfeatAsmcomppathGet()` returns a special pointer `ProAsmcomppath` from a variant feature. `ProAsmcomppath` is the pointer to the component path from the model owner of the top level inheritance feature or the top flexible component to the specified variant feature model.

Both these pointers can be used for accessing properties that can be modified by the presence of an inheritance feature or flexible component. Other functions will return an error `PRO_TK_INVALID_PTR` if provided with this pointer.

Note

If you are using the variant feature model handle and you need to use an unsupported function on it, you can attempt to retrieve the actual model by extracting the model name and type from the model and using `ProMdlnameRetrieve()` to get the original model. This requires that the parent model be accessible in session. Remember that the parent model will not reflect variations applied by the variant feature.

Use the function `ProMdlIsVariantfeatMdl()` to identify if a model pointer is a variant feature model handle.

Accessing Properties of Variant Features

The following items are supported by functions that accept the variant feature model handle. Except where noted, both read and write access is supported

- Dimensions
  - Variant dimensions of the inheritance feature:
    - Value
    - Tolerance
    - Dimension bound

        - Nonvariant dimensions of the inheritance feature (Properties are read only)
- Parameters
    - Variant parameters of the inheritance feature:
        - Value
        - Nonvariant parameters of the inheritance feature (Properties are read only)
- Feature (Inheritance feature only)
    - Variant member features:
        - Suppressed or resumed or erased status
        - Nonvariant member features (Properties are read only)
        - Replaced or alternate feature references
- Annotation (Inheritance feature only)
    - Variant geometric tolerances:
        - Copy status
        - Value
        - Nonvariant geometric tolerances (Properties are read only)
        - Variant notes
        - Copy status
        - Shown or hidden property
        - Nonvariant notes (Properties are read only)
        - Variant symbols
        - Copy status
        - Nonvariant symbols (Properties are read only)
        - Variant surface finishes
        - Copy status
        - Value
        - Nonvariant surface finishes (Properties are read only)

Read Functions Supporting Inheritance Features

Basic Model Properties	`ProMdlMdlnameGet()`
	`ProMdlTypeGet()`
	`ProMdlSubtypeGet()`
	`ProMdlPrincipalunitsystemGet()`
	`ProUnitsystemUnitGet()`
	`ProUnitsystemTypeGet()`
	`ProUnitsystemIsStandard()`
	`ProUnitIsStandard()`
	`ProUnitTypeGet()`
	`ProUnitConversionGet()`
	`ProSolidAccuracyGet()`
	`ProSolidOutlineGet()`
	`ProSolidOutlineCompute()`

Basic Model Items	`ProModelitemNameGet()`
	`ProModelitemDefaultnameGet()`
	`ProSelectionModelitemGet()`

Dimensions	`ProSolidDimensionVisit()`
	`ProFeatureDimensionVisit()`
	`ProDimensionNomvalueGet()`
	`ProDimensionBoundGet()`
	`ProDimensionSymtextGet()`
	`ProDimensionTollabelGet()`
	`ProDimensionSymbolGet()`
	`ProDimensionValueGet()`
	`ProDimensionToltypeGet()`
	`ProDimensionToleranceGet()`
	`ProDimensionTolerancedecimalsGet()`
	`ProDimensionTolerancedenominatorGet()`
	`ProDimensionTypeGet()`
	`ProDimensionIsFractional()`
	`ProDimensionDecimalsGet()`
	`ProDimensionDenominatorGet()`

	ProDimensionIsReldriven() ProDimensionIsRegenednegative() ProDimensionTextGet() ProDimensionTextstyleGet() ProDimensionIsToleranceDisplayed() ProDimensionIsBasic() ProDimensionIsInspection() ProDimensionIsBaseline() ProDimensionIsOrdinate() ProDimensionOrdinatestandardGet() ProDimensionLocationGet() ProDimensionPlaneGet() ProDimensionAttachmentsGet() ProDimensionOverridevalueGet() ProDimensionValuedisplayGet()
Parameters	ProParameterVisit() ProParameterValueWithUnitsGet() ProParameterUnitsGet() ProParameterIsEnumerated() ProParameterRangeGet() ProParameterScaledvalueGet()
Features	ProSolidFeatVisit() ProFeatureTypeGet() ProFeatureTypenameGet() ProFeatureSubtypeGet() ProFeatureStatusGet() ProSolidFeatstatusGet() ProFeatureChildrenGet() ProFeatureParentsGet() ProFeatureCopyinfoGet() ProFeatureGroupGet() ProFeatureGroupStatusGet()

	ProFeatureGrppatternStatusGet() ProFeatureHasGeomchks() ProFeatureIsIncomplete() ProFeatureIsNcseq() ProFeatureIsReadonly() ProFeatureNumSectionsGet() ProFeatureNumberGet() ProFeaturePatternGet() ProFeaturePatternStatusGet() ProFeatureSectionCopy() ProFeatureSolidGet() ProFeatureVerstampGet() ProFeatureVisibilityGet()
Annotations	ProMdlNoteVisit() ProSolidDispoutlineGet() ProNoteTextGet() ProNotePlacementGet() ProNoteURLGet() ProNoteOwnerGet() ProNoteTextstyleGet() ProTextStyle*Get() ProNoteDtlnoteGet()

Annotations	ProNoteLeaderstyleGet() ProNoteElbowlengthGet() ProMdlGtolVisit() ProGtolTopModelGet() ProGtolTypeGet() ProGtol*Get() ProGtolRightTextGet() ProGtolTopTextGet() ProGtolPrefixGet() ProGtolSuffixGet() ProAnnotationTextstyleGet() ProGtoltextTextstyleGet() ProSolidDtlsyminstVisit() ProDtlsyminstDataGet() ProSolidSurffinishVisit() ProSurffinishValueGet() ProSurffinishReferencesGet() ProSurffinishDataGet() ProAnnotationIsShown() ProAnnotationIsInactive() ProAnnotationElementGet() ProAnnotationelemFeatureGet() ProAnnotationplaneAngleGet() ProAnnotationplaneFrozenGet() ProAnnotationplanePlaneGet() ProAnnotationplaneReferenceGet() ProAnnotationplaneTypeGet() ProAnnotationplaneVectorGet() ProAnnotationplaneViewnameGet() ProGtolDatumReferencesGet()

Write and Modification Functions Supporting Inheritance Features

Dimensions	`ProDimensionValueSet()`
	`ProDimensionToleranceSet()`
	`ProDimensionBoundSet()`

Parameters	`ProParameterValueWithUnitsSet()`

Features	`ProFeatureSuppress()`
	`ProFeatureResume()`
	`ProFeatureDelete()`

Annotations	`ProNoteTextSet()`
	`ProNoteURLSet()`
	`ProNoteElbowlengthSet()`
	`ProNoteTextstyleSet()`
	`ProTextStyle*Set()`
	`ProGtolValueStringSet()`
	`ProGtolPrefixSet()`
	`ProGtolSuffixSet()`
	`ProGtolTopTextSet()`
	`ProGtolRightTextSet()`
	`ProAnnotationTextstyleSet()`
	`ProGtoltextTextstyleSet()`
	`ProAnnotationShow()`
	`ProCombstateAnnotationErase()`
	`ProSurffinishValueSet()`
	`ProGtolDatumReferencesSet()`
	`ProGtolDatumReferencesSet()`

Variant Model Items

Function Introduced:

- **ProVariantfeatItemsVisit()**
- **ProVariantfeatItemStandardize()**
- **ProVariantfeatItemCopyGet()**

- **ProVariantfeatItemCopySet()**
- **ProVariantfeatItemCopyUnset()**

Use the function `ProVariantfeatItemsVisit()` to visit the variant items (dimensions, features, and annotations) owned by an inheritance feature or flexible component. The item handles will contain the variant feature model pointer.

Use the function `ProVariantfeatItemStandardize()` to remove a varied item from the inheritance feature or flexible component.

Use the function `ProVariantfeatItemCopyGet()` to obtain the `copy` flag for a given item in the inheritance feature or flexible component. The input arguments of this function are:

- *feature*—Specifies the variant feature
- *item*—Specifies the item.

The output arguments of this function are:

- *copy*—Specifies whether or not to copy the item into the variant feature. If this argument returns true, the item is copied into the feature. If it is false, the item is not copied. This value overrides the value of the features *copy all* flag.

Use the function `ProVariantfeatItemCopySet()` to assign the `copy` flag for a given item in the inheritance feature or flexible component.

Use the function `ProVariantfeatItemCopyUnset()` to remove the `copy` flag for a given item in the inheritance feature or flexible component.

Variant Parameters

Function Introduced:

- **ProVariantfeatParamsVisit()**
- **ProVariantfeatParamStandardize()**

Use the function `ProVariantfeatParamsVisit()` to visit only the variant parameters owned by an inheritance feature or flexible component. The parameter handles contain the variant feature model pointer.

Use the function `ProVariantfeatParamStandardize()` to remove a varied parameter from the inheritance feature or flexible component.

Variant References

By using variant references, you can reroute or replace the references for features in the inheritance feature with new references located inside or outside the inheritance feature. In assemblies, references can be to models other than the target model, as long as the model is within the assembly.

In Creo Parametric TOOLKIT, a variant reference is represented by a `ProVariantRef` handle.

Function Introduced:

- **ProVariantrefAlloc()**
- **ProVariantrefOriginalrefGet()**
- **ProVariantrefOriginalrefSet()**
- **ProVariantrefReplacementrefGet()**
- **ProVariantrefReplacementrefSet()**
- **ProVariantrefFeatidsGet()**
- **ProVariantrefFeatidsSet()**
- **ProVariantfeatVariantrefsGet()**
- **ProVariantfeatVariantrefsSet()**
- **ProVariantrefFree()**
- **ProVariantrefProarrayFree()**

Use the function `ProVariantrefAlloc()` to allocate a handle used to describe a variant reference assigned to a variant feature (like inheritance features). The input arguments of this function are:

- *original_ref*—Specifies the initial reference handle.
- *replacement_ref*—Specifies the replacement reference handle.
- *feat_ids*—Specifies a `ProArray` of feature ids (from the base model) which will be assigned to the reference replacement action.

Use the function `ProVariantrefOriginalrefGet()` to obtain the original reference that is replaced.

Use the function `ProVariantrefOriginalrefSet()` to assign the original reference that is to be replaced.

Use the function `ProVariantrefReplacementrefGet()` to obtain the replacement reference for replacing the original reference in this variant feature.

Use the function `ProVariantrefReplacementrefSet()` to assign the replacement reference for replacing the original reference in this variant feature.

Use the function `ProVariantrefFeatidsGet()` to obtain an array of the feature ids taken from the base model which are assigned the replacement reference.

Use the function `ProVariantrefFeatidsSet()` to assign an array of the feature ids taken from the base model which are assigned the reference replacement.

Note

An assignment of a replacement reference to a feature applies only if the features actually use the replacement reference.

Use the function `ProVariantfeatVariantrefsGet()` to obtain the variant reference assignments (a `ProArray` of the variant reference) stored by this variant feature.

Use the function `ProVariantfeatVariantrefsSet()` to assign the variant reference assignments stored by this variant feature.

Use the function `ProVariantrefFree()` to free a handle used to describe a variant reference assigned to a variant feature and the function `ProVariantrefProarrayFree()` to free an array of handles used to describe a variant reference assigned to a variant feature.

59

Drawings

This chapter describes the Creo Parametric TOOLKIT functions that deal with drawings. Unless otherwise specified, functions that operate on drawings use screen coordinates. See the Core: 3D Geometry on page 170 chapter to find out more about screen coordinates and how to convert to drawing coordinates (or paper coordinates).

Creating Drawings from Templates

Function Introduced:

- **ProDrawingFromTmpltCreate()**
- **ProDrawingFromTemplateCreate()**

Note

The function `ProDrawingFromTmpltCreate()` will be deprecated in a future release. Use the function `ProDrawingFromTemplateCreate()` instead.

Use of drawing templates simplifies drawing creation. Such templates contain drawing views with various properties such as:

- Cross section view
- Simplified representation
- Dimensions On or Off
- Repeat regions (tables based on BOM balloons)

Use the function `ProDrawingFromTemplateCreate()` to create a drawing from a template and to return a structure containing any errors encountered during drawing creation. This input arguments are:

- New drawing name
- Name of existing template to use
- Solid model to use when creating drawing

Note

In the function `ProDrawingFromTemplateCreate()`, the object `ProMdlnameShortdata` supports a file name of 31 characters or less for a model.

- Drawing output options that specify how you want to view drawings output. Chose any or all from the following list:
  - `PRODWGCREATE_DISPLAY_DRAWING`—display new drawing in a window
  - `PRODWGCREATE_SHOW_ERROR_DIALOG`—display the template error dialog to the user

- PRODWGCREATE_WRITE_ERRORS_TO_FILE—write the errors to a disk file

The function returns an error structure if any errors occur. The error structure contains an array of errors. Each error message may have:

- Error type
- Name of view where error occurred
- Name of drawing sheet where error occurred
- Name of the invalid or missing object

If the template and/ or the model name of the drawing is an embedded model name, the function ProDrawingFromTemplateCreate() returns the error PRO_TK_INVALID_NAME.

If there are one or more errors while creating the drawing, the function returns the error PRO_TK_DWGCREATE_ERRORS.

Example 1: Drawing Creation from a Template

The sample code in the file UgDrawingFromTmpltCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing shows how to create a drawing from a template.

Diagnosing Drawing Creation Errors

Functions Introduced:

- **ProDwgcreateErrsFree()**
- **ProDwgcreateErrsCountGet()**
- **ProDwgcreateErrTypeGet()**
- **ProDwgcreateErrViewNameGet()**
- **ProDwgcreateErrSheetGet()**
- **ProDwgcreateErrViewGet()**
- **ProDwgcreateErrObjNameGet()**

The function ProDwgcreateErrsFree() frees an existing errors table.

Use the function ProDwgcreateErrsCountGet() to return the number of drawing creation errors in the table.

The function ProDwgcreateErrTypeGet() returns the type of a drawing creation error.

Use the function ProDwgcreateErrViewNameGet() returns the name of the template view where the error occurred.

The function `ProDwgcreateErrSheetGet()` returns the drawing sheet number where the error occurred.

Use the function `ProDwgcreateErrViewGet()` returns the drawing view where the error occurred. This function is valid for the following error types:

- `PRODWGCRTERR_EXPLODE_DOESNT_EXIST`
- `PRODWGCRTERR_MODEL_NOT_EXPLODABLE`
- `PRODWGCRTERR_SEC_NOT_PERP`
- `PRODWGCRTERR_NO_RPT_REGIONS`
- `PRODWGCRTERR_FIRST_REGION_USED`
- `PRODWGCRTERR_NOT_PROCESS_ASSEM`
- `PRODWGCRTERR_TEMPLATE_USED`
- `PRODWGCRTERR_SEC_NOT_PARALLEL`
- `PRODWGCRTERR_SIMP_REP_DOESNT_EXIST`

`ProDwgcreateErrObjNameGet()` returns the name of the model invalid. This function is valid for the following error types:

- `PRODWGCRTERR_SAVED_VIEW_DOESNT_EXIST`
- `PRODWGCRTERR_X_SEC_DOESNT_EXIST`
- `PRODWGCRTERR_EXPLODE_DOESNT_EXIST`
- `PRODWGCRTERR_SEC_NOT_PERP`
- `PRODWGCRTERR_SEC_NOT_PARALLEL`
- `PRODWGCRTERR_SIMP_REP_DOESNT_EXIST`

Drawing Setup

Functions Introduced:

- **ProInputFileRead()**
- **ProOutputFileMdlnameWrite()**
- **ProDrawingSetupOptionGet()**
- **ProDrawingSetupOptionSet()**
- **ProMdlDetailOptionGet()**
- **ProMdlDetailOptionSet()**

You can set all drawing setup file options from a Creo Parametric TOOLKIT application. To do this, import a text file in the format of the drawing setup file using the function `ProInputFileRead()`, with the file type set to `PRO_DWG_`

SETUP_FILE. You can create such a file from Creo Parametric TOOLKIT with function ProOutputFileMdlnameWrite(). See the Interface: Data Exchange on page 664 chapter for information on these functions.

Use the functions ProDrawingSetupOptionGet() and ProDrawingSetupOptionSet() to return and assign a specific drawing setup file option.

Some of the 2D detail options of the drawings are also applicable to 3D models. The detail options are applied to the model in the same way as to the drawings. Further, the detail options for the model and their values are also stored in the model in the same way as in the drawings.

Use the function ProMdlDetailOptionGet() to get the value of the specified detail option for the model. The function ProMdlDetailOptionSet() sets the value for the specified detail option of the model.

Context in Drawing Mode

Functions Introduced:

- **ProRibbonContextGet()**
- **ProRibbonContextSet()**

The functions ProRibbonContextGet() and ProRibbonContextSet() retrieve and set the context for the specified drawing window, respectively.

From Pro/ENGINEER Wildfire 5.0 onwards, you can specify a context for a drawing. A context presents a set of commands relevant to the type of task that you are currently performing on a drawing. The contexts in the Drawing mode are **Layout**, **Table**, **Annotate**, **Sketch**, **AutobuildZ**, **Legacy Migration**, **Analysis**, and **Review**.

Note

To set the context to **AutobuildZ**, you must first set the configuration option autobuildz_enabled to yes.

One of the arguments to the ProRibbonContextGet() and ProRibbonContextSet() functions is *context*, which is of the type ProRibbonContext. ProRibbonContext is an enumerated type, which takes the following values:

- PRO_RBNCONTEXT_DWG_NONE
- PRO_RBNCONTEXT_DWG_LAYOUT

- PRO_RBNCONTEXT_DWG_TABLE
- PRO_RBNCONTEXT_DWG_ANNOTATE
- PRO_RBNCONTEXT_DWG_SKETCH
- PRO_RBNCONTEXT_DWG_AUTOBUILDZ
- PRO_RBNCONTEXT_DWG_REVIEW
- PRO_RBNCONTEXT_DWG_PUBLISH

Note

If you set *context* to PRO_RBNCONTEXT_DWG_NONE, all actions for the selected object are available in all contexts, except the contexts in which selection is not allowed. For example, the publish context.

Access Drawing Location in Grid

Function Introduced:

- **ProDrawingPosToLocgrid()**

Use the function ProDrawingPosToLocgrid() to find the grid coordinates of a location in the specified drawing. The function specifies the position of a point, expressed in screen coordinates. ProDrawingPosToLocgrid() returns strings representing the row and column containing the point.

Drawing Tree

Pro/ENGINEER Wildfire 5.0 onward, you can view a hierarchical representation of drawing items in an active drawing sheet. This representation is called a **Drawing Tree**. A drawing tree facilitates selection and availability of drawing operations for the represented drawing items. For more information, see the Creo Parametric Help. Use the following functions to refresh, expand, and collapse the drawing tree:

Functions Introduced:

- **ProDrawingtreeRefresh()**
- **ProDrawingtreeExpand()**
- **ProDrawingtreeCollapse()**

Use the function ProDrawingtreeRefresh() to rebuild the drawing tree in the Creo Parametric window that contains the drawing. You can use this function after adding a single drawing item or multiple drawing items to a drawing.

Use the function `ProDrawingtreeExpand()` to expand the drawing tree in the Creo Parametric window and make all drawing sheets and drawing items in the active drawing sheet visible.

Use the function `ProDrawingtreeCollapse()` to collapse all nodes of the drawing tree in the Creo Parametric window and make its child nodes invisible.

The input arguments to these functions are:

- *drawing*—Handle to the drawing that contains the drawing tree.
- *window_id* —ID of the Creo Parametric window in which you want to refresh, expand, or collapse the drawing tree.

Note

Use `PRO_VALUE_UNUSED` to refresh, expand, or collapse the drawing tree in the active window.

Example 2: Performing Operations on a Drawing Tree

The sample code in the file `UgDrwTree.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dwg` shows how to expand, collapse, and refresh the drawing tree in the selected drawing.

Merge Drawings

Function Introduced:

- **ProDrawingMerge()**

Use the function `ProDrawingMerge()` to merge two drawings.

Drawing Sheets

Functions Introduced:

- **ProDrawingSheetNameGet()**
- **ProDrawingSheetsCount()**
- **ProDrawingSheetTrfGet()**
- **ProDrawingSheetSizeGet()**
- **ProDrawingSheetOrientationGet()**
- **ProDrawingSheetUnitsGet()**

- **ProDrawingCurrentSheetGet()**
- **ProDrawingCurrentSheetSet()**
- **ProDrawingSheetCreate()**
- **ProDrawingSheetDelete()**
- **ProDrawingSheetCopy()**
- **ProDrawingSheetsReorder()**
- **ProDrawingSheetFormatIsBlanked()**
- **ProDrawingFormatGet()**
- **ProDrawingFormatAdd()**
- **ProDrawingSheetFormatShow()**
- **ProDrawingSheetFormatHide()**
- **ProDrawingSheetFormatIsShown()**
- **ProDrawingSheetFromFormatGet()**
- **ProDrawingToleranceStandardGet()**
- **ProDrawingToleranceStandardSet()**
- **ProDwgSheetRegenerate()**

Drawing sheets are identified in Creo Parametric TOOLKIT by the same sheet numbers the Creo Parametric user sees.

The function `ProDrawingSheetNameGet()` retrieves the name of the specified drawing sheet.

The function `ProDrawingSheetsCount()` outputs the current number of sheets in the specified drawing.

The function `ProDrawingSheetTrfGet()` provides the matrix that transforms from screen to drawing units or vice versa. (See the Core: Coordinate Systems and Transformations on page 222 chapter for more details on transformations.)

The function `ProDrawingSheetSizeGet()` returns the size of the sheet used for the drawing. If the sheet is a standard paper size, this will be returned, along with the width and height of the sheet measured in the sheet units.

The function `ProDrawingSheetOrientationGet()` returns the orientation of the drawing sheet. The orientation can be either portrait or landscape.

The function `ProDrawingSheetUnitsGet()` retrieves the name of the length units used for measuring the drawing sheet.

The function `ProDrawingCurrentSheetGet()` returns the sheet number of the currently selected sheet in the specified drawing. It returns a positive integer for the retrieved sheet number and a zero for an invalid input argument.

The function `ProDrawingCurrentSheetSet()` sets the sheet number for the currently selected sheet in the specified drawing. One of the input arguments to this function is an integer, *current_sheet*.

The function `ProDrawingSheetCreate()` adds a sheet to a drawing model of type Notebook, Format, Drawing, Diagram, or Report. The output argument to this function is an integer, *new_sheet*. If the drawing sheet is successfully created, *new_sheet* takes the value of the sheet number for the newly created drawing sheet. Otherwise it takes the value, -1.

The function `ProDrawingSheetDelete()` deletes a sheet from a drawing model of type Notebook, Format, Drawing, Diagram, or Report. You need to enter the type of the drawing model and the sheet number of the sheet to be deleted as input arguments.

The function `ProDrawingSheetCopy()` creates a copy of a specified drawing sheet. The input arguments to this function are:

- *drawing*—The drawing model handle. Set it to `NULL` to copy a sheet from the current drawing.
- *sheet_to_copy*—The sheet number of the sheet to be copied. Set it to a value less than 1 to create a copy of the currently selected sheet.

The output argument to this function is:

- *sheet_copy* (optional)—The sheet number of the newly created sheet. Set it to `NULL`, if you do not want the new sheet number.

The function `ProDrawingSheetsReorder()` assigns a new sheet number to a specified sheet, and renumbers the remaining sheets accordingly.

The function `ProDrawingSheetFormatIsBlanked()` identifies whether the format of a specified drawing sheet is blank. In case of the current sheet, set the input arguments *drawing* and *sheet* to `NULL` and to a value less than 1, respectively.

The function `ProDrawingFormatGet()` returns the name of the drawing format that was used for the specified sheet. `ProDrawingFormatAdd()` adds or replaces a specified format into a specified drawing sheet.

The function `ProDrawingSheetFormatShow()` displays the drawing format for a specified drawing sheet. To display the drawing format of the current sheet, set the input arguments *drawing* and *sheet* to `NULL` and to a value less than 1, respectively.

Use the function `ProDrawingSheetFormatHide()` to hide the drawing format for a specified drawing sheet.

The function `ProDrawingSheetFormatIsShown()` identifies if the drawing format for a specified drawing sheet is shown. It returns `PRO_B_TRUE` if the drawing format is shown and returns `PRO_B_FALSE` if the drawing format is not shown.

The function `ProDrawingSheetFromFormatGet()` retrieves the sheet number of the drawing format for a specified drawing and sheet number within the drawing.

The function `ProDrawingToleranceStandardGet()` returns the tolerance standard that is assigned to the specified drawing. Use the function `ProDrawingToleranceStandardSet()` to set the tolerance standard for a drawing.

The function `ProDwgSheetRegenerate()` regenerates a specified drawing sheet.

Example 3: Listing Drawing Sheets

The sample code in the file `UgDrawingFromTmpltCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing` shows how to list the sheets in the current drawing.

Example 4: Creating a Copy of the Current Drawing Sheet

The sample code in the file `UgSheetCopy.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dwg` shows how to Add a drawing sheet to the current drawing-type object.

Drawing Format Files

The format of a drawing refers to the boundary lines, referencing marks and graphic elements that appear on every sheet before any drawing elements are shown or added. These usually include items such as tables for the company name, detailers name, revision number and date. In a Creo Parametric drawing, you can associate a format file (`.frm`) with the drawing. This file carries all the format graphical information, and it can also carry some optional default attributes like text size and draft scale. The functions described in this section allow you to get and set the size of the drawing format.

Functions Introduced:

- **ProDrawingFormatSizeSet()**
- **ProDrawingFormatSizeGet()**

The function `ProDrawingFormatSizeSet()` sets the size of the drawing format in the specified drawing. You can add a standard or customize size format in the drawing. The input arguments are:

- *drawing*—Specifies the name of the drawing.
- *drawing_sheet*—Specifies the number of the drawing sheet where the drawing format must be set.
- *size*—Specifies the size of the drawing using the enumerated data type `ProPlotPaperSize`.
- *width*—Specifies the width of the drawing in inches, when *size* is set to `VARIABLE_SIZE_PLOT`.

  It specifies the width of the drawing in millimeters, when *size* is set to `VARIABLE_SIZE_IN_MM_PLOT`.

  Note

  This argument is ignored for all the other sizes of the drawing except `VARIABLE_SIZE_PLOT` and `VARIABLE_SIZE_IN_MM_PLOT`. In such cases specify the argument as `PRO_VALUE_UNUSED`.

- *height*—Specifies the height of the drawing in inches, when the *size* is set to `VARIABLE_SIZE_PLOT`.

  It specifies the width of the drawing in millimeters, when the *size* is set to `VARIABLE_SIZE_IN_MM_PLOT`.

  Note

  This argument is ignored for all the other sizes of the drawing except `VARIABLE_SIZE_PLOT` and `VARIABLE_SIZE_IN_MM_PLOT`. In such cases specify the argument as `PRO_VALUE_UNUSED`.

Use the function `ProDrawingFormatSizeGet()` to get the size of the drawing format in the specified drawing. The function returns the size of the drawing, and the width and height of the drawing in inches.

Drawing Views and Models

Drawing views are identified by the OHandle `ProView`. This is the same object handle used to reference 3D model views, accessed by the functions in `ProView.h`, but there are no cases where the same object can be accessed by

both types of function. The general rule is that functions for 3D views are in `ProView.h`, and start with `ProView`; functions to manipulate drawing views are in `ProDrawing.h` and therefore always start with `ProDrawing`.

Each drawing view has a solid attached to it. The same solid can appear in more than one view. Some solids may be attached to the drawing, but not appear in any view, if the Creo Parametric user has deleted all views of it without also removing the solid itself from the drawing.

Listing Drawing Views

Functions Introduced:

- **ProDrawingViewsCollect()**
- **ProDrawingViewVisit()**
- **ProDrawingViewSheetGet()**
- **ProDrawingErasedviewSheetGet()**
- **ProDrawingViewOutlineGet()**
- **ProDrawingViewTypeGet()**
- **ProDrawingViewIdGet()**
- **ProDrawingViewInit()**
- **ProDrawingViewonlyOpen()**
- **ProDrawingViewParentGet()**
- **ProDrawingViewChildrenGet()**
- **ProDrawingViewOriginGet()**
- **ProDrawingViewAlignmentGet()**
- **ProDrawingViewScaleIsUserdefined()**
- **ProDrawingViewScaleGet()**
- **ProDrawingScaleGet()**
- **ProDrawingViewPerspectiveScaleGet()**
- **ProDrawingViewFlagGet()**
- **ProDrawingViewDisplayGet()**
- **ProDrawingViewColorSourceGet()**
- **ProDrawingViewSolidGet()**
- **ProDrawingViewTransformGet()**
- **ProDrawingViewNameGet()**
- **ProDrawingViewZclippingGet()**

- **ProDrawingViewDatumdisplayGet()**
- **ProDrawingViewPipingdisplayGet()**
- **ProDrawingViewIsErased()**
- **ProDrawingViewNeedsRegen()**

`ProDrawingViewsCollect()` outputs an array of `ProView` handles to all the views in a drawing. `ProDrawingViewVisit()` is an alternative way to find the views, and conforms to the usual form of visit functions. `ProDrawingViewSheetGet()` outputs the number of the sheet in which a specified view appears.

The function `ProDrawingErasedviewSheetGet()` outputs the number of the sheet, which contained the view that was erased. If the sheet that contained the erased view is deleted, the sheet ID is returned as `PRO_VALUE_UNUSED`.

The function `ProDrawingViewOutlineGet()` outputs the position of the view in the sheet.

The function `ProDrawingViewTypeGet()` returns the type of a specified drawing view. A drawing view can be of the following types:

- `PRO_VIEW_GENERAL`—Specifies a general drawing view.
- `PRO_VIEW_PROJECTION`—Specifies a projected drawing view.
- `PRO_VIEW_AUXILIARY`—Specifies an auxiliary drawing view.
- `PRO_VIEW_DETAIL`—Specifies a detailed drawing view.
- `PRO_VIEW_REVOLVE`—Specifies a revolved drawing view.
- `PRO_VIEW_COPY_AND_ALIGN`—Specifies a copy and align drawing view.
- `PRO_VIEW_OF_FLAT_TYPE`—Specifies a flat type drawing view.

The function `ProDrawingViewIdGet()` retrieves the ID of the drawing view specified by the drawing and the drawing view handles.

The function `ProDrawingViewInit()` retrieves the drawing view handle based on the specified drawing handle and the view ID.

The function `ProDrawingViewonlyOpen()` opens a drawing in the view only mode.

The function `ProDrawingViewParentGet()` returns the parent view of a specified drawing view. The function `ProDrawingViewChildrenGet()` returns the child views of a drawing view and the total number of children.

`ProDrawingViewOriginGet()` returns the location of the origin in terms of the `ProPoint3d` object and the selection reference in terms of the `ProSelection` object for a specified drawing view.

ProDrawingViewAlignmentGet() returns the alignment of a drawing view with respect to another view. It returns the alignment style of the drawing view that can be either horizontal or vertical, the reference view to which the view is aligned, and the alignment references of both the aligned and reference views.

ProDrawingViewScaleIsUserdefined() returns a boolean value depending on whether the drawing has a user-assigned scale or not.

ProDrawingViewScaleGet() returns the scale factor applied, even if the view is not a scaled view. ProDrawingScaleGet() returns the overall drawing scale that is applied to all unscaled views.

The function ProDrawingViewPerspectiveScaleGet() returns the perspective scale applied to a drawing view. This scale option is available only for general views. The function returns the following output arguments:

- *eye_dist*—Specifies the eye-point distance from model space.
- *view_dia*—Specifies the view diameter in paper units such as mm.

The function ProDrawingViewFlagGet() identifies if the projection arrow flag has been set for a projected or detailed drawing view.

ProDrawingViewDisplayGet() outputs the ProDrawingViewDisplay structure that describes the display status of the drawing view. The fields in the structure, all either enums or booleans, are listed below:

- style—Whether wireframe, hidden line, or shaded.
- quilt_hlr—Whether hidden-line-removal is applied to quilts.
- tangent_edge_display—Style of line used for tangent edges.
- cable_display—Whether cables are shown by centerline, as thick, or using the current default.
- concept_model—Whether the skeleton is displayed.
- weld_xsec—Whether welds are included in the cross-section.

ProDrawingViewColorSourceGet() returns the color source of the drawing view representation. The color source can be of the following types:

- PRO_VIEW_MODEL_COLOR—Specifies that the drawing colors are determined by the model settings.
- PRO_VIEW_DRAWING_COLOR—Specifies that the drawing colors are determined by the drawings settings.

ProDrawingViewSolidGet() provides the handle to the solid that is being displayed in the drawing view. ProDrawingViewTransformGet() returns the matrix that describes the transform between 3D solid coordinates and 2D screen coordinates for a specified view.

ProDrawingViewNameGet() returns the name of a specified view in the drawing.

`ProDrawingViewZclippingGet()` returns the reference of the Z-clipping on the drawing view. The reference can be an edge, datum, or point on the surface that is parallel to the view. Geometry contained in the Z-clipping plane and in front of the plane appears, but geometry behind the plane does not appear. The system clips geometry that intersects the plane.

The function `ProDrawingViewDatumdisplayGet()` determines if a solid model datum has been explicitly shown in a particular drawing view.

The function `ProDrawingViewPipingdisplayGet()` returns the piping display option for a drawing view.

The function `ProDrawingViewIsErased()` identifies if the drawing view is erased or not.

The function `ProDrawingViewNeedsRegen()` identifies whether the drawing or the specified drawing view needs to be regenerated.

Example 5: Listing the Views in a Drawing

The sample code in the file `UgDrawingViews.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing` shows a command that creates an information window reporting information about all the views in a drawing.

Modifying Views

Functions Introduced:

- **ProDrawingViewMove()**
- **ProDrawingViewDelete()**
- **ProDrawingViewErase()**
- **ProDrawingViewResume()**
- **ProDrawingViewOriginSet()**
- **ProDrawingViewAlignmentSet()**
- **ProDrawingViewScaleSet()**
- **ProDrawingScaleSet()**
- **ProDrawingViewProjectionSet()**
- **ProDrawingViewFlagSet()**
- **ProDrawingViewDisplaySet()**
- **ProDrawingViewNameSet()**
- **ProDrawingViewZclippingSet()**
- **ProDrawingViewPipingdisplaySet()**

- **ProDrawingViewRegenerate()**
- **ProDwgViewRegenerate()**

The function `ProDrawingViewMove()` moves the specified drawing view and its child views by a vector to a new position in screen coordinates. This function performs the same operation as the Creo Parametric command **Sketch ► Edit ► Move Special**.

The function `ProDrawingViewDelete()` deletes the drawing view and any of its child views, or fails if the view has any children.

The function `ProDrawingViewErase()` erases a specified drawing view.

The function `ProDrawingViewResume()` resumes the specified drawing view.

`ProDrawingViewOriginSet()` assigns the location of the origin in terms of the `ProPoint3d` object and the selection reference in terms of the `ProSelection` object for a specified drawing view.

The function `ProDrawingViewAlignmentSet()` assigns the alignment of a drawing view with respect to another view. This function assigns the following input arguments related to alignment:

- *view_reference*—Specifies the reference view to which the drawing view is aligned.
- *align_style*—Specifies the alignment style in terms of the enumerated type `ProDrawingViewAlignStyle`. The alignment style can be either horizontal or vertical. In case of horizontal alignment, the drawing view and the view it is aligned to lie on the same horizontal line. In case of vertical alignment, the drawing view and the view it is aligned to lie on the same vertical line.
- *align_ref_1*—Specifies the alignment reference of the referenced view. If this is set to `NULL`, the reference view is aligned according to its view origin.
- *align_ref_2*—Specifies the alignment reference of the aligned view. If this is set to `NULL`, the aligned view is aligned according to its view origin.

The function `ProDrawingViewScaleSet()` modifies the scale of a scaled view; `ProDrawingScaleSet()` modifies the overall drawing scale that is applied to all unscaled views.

The function `ProDrawingViewProjectionSet()` assigns the specified drawing view as a projection.

The function `ProDrawingViewFlagSet()` sets the projection arrow flag to `TRUE` for a projected or detailed drawing view.

The function `ProDrawingViewDisplaySet()` assigns the fields in the `ProDrawingViewDisplay` structure that describe the display status of the drawing view. This function does not repaint the drawing view, use the function `ProWindowRepaint()` to repaint the view.

Note

In order to modify the `concept_model` field in the `ProDrawingViewDisplay` structure, you require an Assembly license.

The function `ProDrawingViewNameSet()` assigns a name to a specified drawing view.

Note

The configuration option `allow_duplicate_view_names` enables you to set duplicate names for drawing views. When the configuration option is set to `no`, the function returns an error if another drawing view in the specified drawing exists with the same name.

The function `ProDrawingViewZclippingSet()` sets the Z-clipping on the drawing view to reference a given edge, datum, or point on the surface that is parallel to the view. Geometry contained in the Z-clipping plane and in front of the plane appears, but geometry behind the plane does not appear. The system clips geometry that intersects the plane.

The function `ProDrawingViewPipingdisplaySet()` assigns the piping display option for a drawing view. The piping display options are as follows:

- `PRO_PIPINGDISP_DEFAULT`—Displays the default appearance of pipes for the piping assembly.
- `PRO_PIPINGDISP_CENTERLINE`—Displays pipes as centerlines without insulation.
- `PRO_PIPINGDISP_THICK_PIPES`—Displays thick pipes without insulation.
- `PRO_PIPINGDISP_THICK_PIPES_AND_INSULATION`—Displays thick pipes and insulation.

The function `ProDrawingViewRegenerate()` regenerates the drawing view specified by the `ProView` view handle.

The function `ProDwgViewRegenerate()` erases the displayed view of the current object, regenerates the view from the current drawing, then redisplays the view.

Creating Views

Function Introduced:

- **ProDrawingGeneralviewCreate()**
- **ProDrawingProjectedviewCreate()**

A general view is usually the first view placed on a drawing sheet. It is the most versatile view, that is, it can be scaled or rotated to any setting.

A projected view is an orthographic projection of another view's geometry along a horizontal or vertical direction.

The above functions create general and projected drawing views.

Example 6: Creating General and Projected Drawing Views

The sample code in the file `UgDrawingViews.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing` adds a new sheet to a drawing, and creates one general and two projected views of a selected model.

Background Views

Functions Introduced:

- **ProDrawingViewIsBackground()**
- **ProDrawingBackgroundViewGet()**

Views can be assigned not only to the solid views that the user creates, but also to each sheet. The view assigned to a drawing sheet is called a background view. Function `ProDrawingViewIsBackground()` determines whether the specified view is a background view.

If you use `ProSelect()` for an item in a drawing that is not inside a solid view, such as a detail item, the `ProView` view handle output by `ProSelectionViewGet()` is the background view.

Note

These functions supersede the functions `ProDrawingViewIsOverlay()` and `ProDrawingOverlayviewGet()`, which previously served the same purpose.

Detailed Views

Functions Introduced:

- **ProDrawingViewDetailCreate()**
- **ProDrawingViewDetailReferenceGet()**
- **ProDrawingViewDetailReferenceSet()**
- **ProDrawingViewDetailCurvedataGet()**
- **ProDrawingViewDetailCurvedataSet()**
- **ProDrawingViewDetailBoundaryGet()**
- **ProDrawingViewDetailBoundarySet()**

A detailed view is a small portion of a model shown enlarged in another view.

The function `ProDrawingViewDetailCreate()` creates a detailed view given the reference point on the parent view, the spline curve data, and location of the new view. A note with the detailed view name and the spline curve border are included in the parent view for the created detailed view.

The functions `ProDrawingViewDetailReferenceGet()` and `ProDrawingViewDetailReferenceSet()` return and assign, respectively, the reference point on the parent view for a specified detailed view.

The functions `ProDrawingViewDetailCurvedataGet()` and `ProDrawingViewDetailCurvedataSet()` retrieve and assign, respectively, the spline curve data in terms of the `ProCurvedata` handle for a specified detailed view. The output argument, `curve_data` specifies the following:

- The X and Y coordinate directions match the screen space.
- The coordinate point (0,0) maps to the reference point.
- The scaling unit is of one inch relative to the top model of the view. If two points in the spline are at a distance of '1' from each other, then in the actual view, the points will be one inch distant from each other, if measured in the scale of the top model. For example, if one of the points in the spline definition has coordinates (0.5, 0.0, 0.0), then the position of that point is not half an inch to the right of the reference point on the paper. Instead, when projected as a point in the space of the top model of the view, it is half an inch to the right of the reference point when measured in the space of that model.

Use the function `ProCurvedataAlloc()` to allocate memory for the curve data structure.

The function `ProDrawingViewDetailBoundaryGet()` retrieves the type of spline curve used to define the boundary of the detailed view and also identifies whether the boundary is displayed on the parent view.

The function `ProDrawingViewDetailBoundarySet()` assigns the boundary type for a detailed view in terms of the enumerated type `ProViewDetailBoundaryType`. The types of boundaries are:

- PRO_DETAIL_BOUNDARY_CIRCLE—Draws a circle in the parent view.
- PRO_DETAIL_BOUNDARY_ELLIPSE—Draws an ellipse in the parent view.
- PRO_DETAIL_BOUNDARY_HORZ_VER_ELLIPSE—Draws an ellipse with a horizontal or vertical major axis.
- PRO_DETAIL_BOUNDARY_SPLINE—Displays the spline boundary drawn by the user in the parent view.
- PRO_DETAIL_BOUNDARY_ASME_CIRCLE—Displays an ASME standard-compliant circle as an arc with arrows and the detailed view name.

This function also sets the ProBoolean argument *show* to display the boundary of the detailed view in the parent view.

Auxiliary Views

Functions Introduced:

- **ProDrawingViewAuxiliaryCreate()**
- **ProDrawingViewAuxiliarySet()**
- **ProDrawingViewAuxiliaryInfoGet()**

An auxiliary view is a type of projected view that projects at right angles to a selected surface or axis. The selected surface or axis in the parent view must be perpendicular to the plane of the screen.

The function ProDrawingViewAuxiliaryCreate() creates an auxiliary view given the selection reference and the point location.

The function ProDrawingViewAuxiliarySet() sets a specified drawing view as the auxiliary view.

The function ProDrawingViewAuxiliaryInfoGet() retrieves information such as the selection reference in terms of the ProSelection object and the point location in terms of the ProPoint3d object for a specified auxiliary view.

Revolved Views

Functions Introduced:

- **ProDrawingViewRevolveCreate()**
- **ProDrawingViewRevolveInfoGet()**

A revolved view is a cross section of a drawing view revolved 90 degrees around a cutting plane projection.

The function ProDrawingViewRevolveCreate() creates a revolved view given a cross section, the selection reference, and the point location.

The function ProDrawingViewRevolveInfoGet() retrieves information such as the cross section in terms of the ProXsec object, the selection reference in terms of the ProSelection object, and the point location in terms of the ProPoint3d object for a specified revolved view.

Draft Views

Functions Introduced:

- **ProDrawingDraftViewsCollect()**
- **ProDrawingViewIsDraft()**
- **ProDrawingDraftViewCreate ()**

Draft views are created from the **Sketch** tab in a drawing using the selected draft entities. The draft entities added to the view are automatically related to the view. Any pre-existing relationship will be removed, and the entity will be related to the draft view.

The function ProDrawingDraftViewsCollect() collects all draft views in the specified drawing. The output argument views is the list of draft views. The function ProDrawingDraftViewsCollect() allocates memory to the ProView handle. To free the memory call the function ProArrayFree().

The function ProDrawingViewIsDraft() determines whether the specified view is a draft view. The input arguments follow:

- *drawing*—Specify the drawing in which the draft view exists.
- *view* —Specify the view using the ProView handle.

If the specified view is a draft view, the function returns a ProBoolean output argument with the value PRO_B_TRUE. Otherwise, the function returns PRO_B_FALSE.

The function ProDrawingDraftViewCreate() creates a draft view in the specified drawing sheet. The input arguments follow:

- *drawing*—Specify the drawing in which draft view is to be created.
- *entities*—Specify at least one daft entity using the ProDtlentity object. Entities might or might not be related to any view.

The function outputs a pointer to the ProView handle.

View Orientation

Functions Introduced:

- **ProDrawingViewOrientationFromNameSet()**
- **ProDrawingViewOrientationFromReferenceSet()**
- **ProDrawingViewOrientationFromAngleSet()**

The orientation type for a drawing view is given by the enumerated type `ProDrawingViewOrientationType`. The view orientation can be of the following types:

- `PRO_VIEW_ORIENT_NAME`—The drawing view is oriented using saved views from the model.
- `PRO_VIEW_ORIENT_GEOM_REF`—The drawing view is oriented using geometric references from the model.
- `PRO_VIEW_ORIENT_ANGLE`—The drawing view is oriented using angles of selected references or custom angles.

Note

The drawing view must be displayed before applying any orientation to it.

The function `ProDrawingViewOrientationFromNameSet()` assigns the orientation of a specified drawing view according to the following input arguments:

- *mdl_view_name*—Specifies the name of the saved view in the model.
- *orientation_name*—Specifies the name of the user-defined orientation for the saved view.
- *x_angle*—Specifies the X angle in degrees for the user-defined orientation.
- *y_angle*—Specifies the Y angle in degrees for the user-defined orientation.

The function `ProDrawingViewOrientationFromReferenceSet()` assigns the orientation of a specified drawing view according to the following input arguments:

- *ref_name_1*—Specifies the name of the first geometric reference.
- *ref_sel_1*—Specifies the first reference selection on the model in terms of the `ProSelection` object.
- *ref_name_2*—Specifies the name of the second geometric reference.
- *ref_sel_2*—Specifies the second reference selection on the model in terms of the `ProSelection` object.

The function `ProDrawingViewOrientationFromAngleSet()` assigns the orientation of a specified drawing view according to the following input arguments:

- *sel*—Specifies the reference selection in terms of the `ProSelection` object. It can be an axis or `NULL` for other type.
- *angle*—Specifies the angle in degrees with the selected reference.

- *rot_ref_name*—Specifies the name of the rotational angle.
- *index*—Specifies the index of the angle setting.

Visible Areas of Views

Functions Introduced:

- **ProDrawingViewVisibleareaTypeGet()**
- **ProDrawingViewFullVisibleAreaSet()**
- **ProDrawingViewHalfVisibleAreaGet()**
- **ProDrawingViewHalfVisibleAreaSet()**
- **ProDrawingViewPartialVisibleAreaSet()**
- **ProDrawingViewPartialVisibleAreaGet()**
- **ProDrawingViewBrokenVisibleAreaGet()**
- **ProDrawingViewBrokenVisibleAreaSet()**
- **ProDrawingViewBrokenNumberGet()**

As you detail your model, certain portions of the model may be more relevant than others or may be clearer if displayed from a different view point.

The function `ProDrawingViewVisibleareaTypeGet()` retrieves the type of visible area for a specified drawing view in terms of the enumerated type `ProDrawingViewVisibleareaType`. The visible area can be of the following types:

- `PRO_VIEW_FULL_AREA`—The complete drawing view is retained as the visible area.
- `PRO_VIEW_HALF_AREA`—A portion of the model from the view on one side of a cutting plane is removed.
- `PRO_VIEW_PARTIAL_AREA`—A portion of the model in a view within a closed boundary is displayed.
- `PRO_VIEW_BROKEN_AREA`—A portion of the model view from between two or more selected points is removed, and the gap between the remaining two portions is closed within a specified distance.

The function `ProDrawingViewFullVisibleAreaSet()` retains the specified drawing view as the full visible area.

The function `ProDrawingViewHalfVisibleAreaGet()`

The function `ProDrawingViewHalfVisibleAreaSet()`

- *plane_ref*—Specifies the selection reference in terms of the `ProSelection` object that divides the drawing view. The cutting plane can be a planar surface or a datum, but it must be perpendicular to the screen in the new view.
- *keep_side*—Specifies the half side of the model that is to be retained.
- *line_standard*`ProDrawingLineStandardType`
  - `PRO_HVL_NONE`—Specifies no line
  - `PRO_HVL_SOLID`—Specifies a solid line.
  - `PRO_HVL_SYMMETRY`—Specifies a symmetry line.
  - `PRO_HVL_SYMMETRY_ISO`—Specifies an ISO-standard symmetry line.
  - `PRO_HVL_SYMMETRY_ASME`—Specifies an ASME-standard symmetry line

The function `ProDrawingViewPartialVisibleAreaGet()`

The function `ProDrawingViewPartialVisibleAreaSet()` assigns the following arguments to define the partial visible area for a specified drawing view:

- *ref_point*—`ProSelection`
- *curve_data*—`ProCurvedata`
- *show_boundary*`ProBooleanPRO_B_TRUE`

The function `ProDrawingViewBrokenVisibleAreaGet()` retrieves the broken visible area for a specified drawing view.

The function `ProDrawingViewBrokenVisibleAreaSet()` assigns the following arguments to define the broken visible area for a specified drawing view:

- *dir*—Specifies the direction of the broken lines that define the broken area to be removed. The direction is given by the enumerated type `ProViewBrokenDir` and takes the following values:
  - `PRO_VIEW_BROKEN_DIR_HORIZONTAL`—Specifies the horizontal direction.
  - `PRO_VIEW_BROKEN_DIR_VERTICAL`—Specifies the vertical direction.
- *first_sel*—Specifies the selection point in terms of the `ProSelection` object for the first break line.
- *second_sel*—Specifies the selection point in terms of the `ProSelection` object for the second break line.

- *line_style*—Specifies the line style for the broken lines in terms of the enumerated type `ProViewBrokenLineStyle`. It can be one of the following types:
  - `PRO_VIEW_BROKEN_LINE_STRAIGHT`—Specifies a straight broken line.
  - `PRO_VIEW_BROKEN_LINE_SKETCH`—Specifies a random sketch drawn by the user that defines the broken line.
  - `PRO_VIEW_BROKEN_LINE_S_CURVE_OUTLINE`—Specifies a S-curve on the view outline.
  - `PRO_VIEW_BROKEN_LINE_S_CURVE_GEOMETRY`—Specifies a S-curve on geometry.
  - `PRO_VIEW_BROKEN_LINE_HEART_BEAT_OUTLINE`—Specifies a heartbeat type of curve on the view outline.
  - `PRO_VIEW_BROKEN_LINE_HEART_BEAT_GEOMETRY`—Specifies a heartbeat type of curve on the geometry.
- *curve_data*—Specifies the spline curve data in terms of the `ProCurvedata` object when the *line_style* is of the type `PRO_VIEW_BROKEN_LINE_SKETCH`.

The function `ProDrawingViewBrokenNumberGet()` returns the number of breaks defined for the broken visible area. Two broken lines define one break.

Note

A broken visible area can be created only for general and projected view types.

Sections of a View

Functions Introduced:

- **ProDrawingViewSectionTypeGet()**
- **ProDrawingView2DSectionGet()**
- **ProDrawingView2DSectionSet()**
- **ProDrawingView2DSectionNumberGet()**
- **ProDrawingView2DSectionFlip()**
- **ProDrawingView2DSectionFlipGet()**
- **ProDrawingView3DSectionGet()**

- **ProDrawingView3DSectionSet()**
- **ProDrawingViewSinglepartSectionGet()**
- **ProDrawingViewSinglepartSectionSet()**
- **ProDrawingView2DSectionTotalSet()**
- **ProDrawingView2DSectionAreaSet()**

The function `ProDrawingViewSectionTypeGet()` retrieves the section type for a specified drawing view in terms of the enumerated type `ProDrawingViewSectionType`. A section can be of the following types:

- `PRO_VIEW_NO_SECTION`—Specifies no section.
- `PRO_VIEW_TOTAL_SECTION`—Specifies the complete drawing view.
- `PRO_VIEW_AREA_SECTION`—Specifies a 2D cross section.
- `PRO_VIEW_3D_SECTION`—Specifies a 3D cross section.
- `PRO_VIEW_PART_SURF_SECTION`—Specifies a section created out of a solid surface or a datum quilt in the model.

A drawing can have many 2D cross sections defined in it. These cross sections are indexed. The first cross section has its index number set to 0. Depending on which 2D cross section you want to work with, specify the index number.

The function `ProDrawingView2DSectionGet()` retrieves the 2D cross section for a specified drawing view.

The function `ProDrawingView2DSectionSet()` assigns the following arguments to define the 2D cross section for a drawing view:

- *sec_name*—Specifies the name of the 2D cross section.
- *sec_area_type*—Specifies the type of section area. The section area is given by the enumerated type `ProDrawingViewSectionAreaType` and can be of the following types:
  - `PRO_VIEW_SECTION_AREA_FULL`—Sectioning is applied to full drawing view.
  - `PRO_VIEW_SECTION_AREA_HALF`—Sectioning is applied to half drawing view depending upon the inputs for half side.
  - `PRO_VIEW_SECTION_AREA_LOCAL`—Specifies local sectioning.
  - `PRO_VIEW_SECTION_AREA_UNFOLD`—Unfold the drawing view and section it.
  - `PRO_VIEW_SECTION_AREA_ALIGNED`—Sectioning is as per the aligned views.
- *ref_sel*—Specifies the selection reference in terms of the `ProSelection` object.

- *curve_data*—Specifies the spline curve data in terms of the `ProCurvedata` handle.
- *arrow_display_view*—Specifies the drawing view, that is, either the parent or child view, where the section arrow is to be displayed.

> **Note**
>
> For a section area of type `PRO_VIEW_SECTION_AREA_FULL` in the above function, you can pass the input arguments *ref_sel*, *curve_data*, and *arrow_display_view* as `NULL`.

The function `ProDrawingView2DSectionNumberGet()` retrieves the number of 2D cross sections defined for a drawing view.

The function `ProDrawingView2DSectionFlip()` flips the direction of 2D cross section in a drawing view. Specify the index of the 2D cross section that you want to flip.

Use the function `ProDrawingView2DSectionFlipGet()` returns a boolean value that indicates the direction in which the 2D cross section has been clipped. Depending on the type of cross section, the boolean value indicates different direction of clipping as below:

- Planar cross section—The boolean value:
  - `PRO_B_FALSE` indicates that the cross section has been clipped in the direction of the positive normal to the cross section plane.
  - `PRO_B_TRUE` indicates that the cross section has been clipped in the opposite direction of the positive normal.
- Offset cross section—The integer value:
  - `PRO_B_FALSE` indicates that material has been removed from the left of the cross section entities if the viewing direction is from the positive side of the entity plane.
  - `PRO_B_TRUE` indicates that the material has been retained from the left of the cross section entities and rest of the material has been removed.

The function `ProDrawingView3DSectionGet()` retrieves the 3D cross section for a specified drawing view.

The function `ProDrawingView3DSectionSet()` assigns the following arguments to define a 3D cross section for a view:

- *sec_name* —Specifies the name of the 3D cross section.
- *show_x_hatch*—Specifies a `ProBoolean` value that determines whether X-hatching is displayed in the 3D cross-sectional view. Set this argument to `PRO_B_TRUE` to display X-hatching.

The function `ProDrawingViewSinglepartSectionGet()` retrieves the section created out of a solid surface or a datum quilt in the model for a specified drawing view.

The function `ProDrawingViewSinglepartSectionSet()` assigns the reference selection in terms of the `ProSelection` object for the solid surface or datum quilt that is used to create the section in the view.

The functions `ProDrawingView2DSectionTotalSet()` and `ProDrawingView2DSectionAreaSet()` enable you to set the visibility of model edges in a 2D cross section view of a drawing.

The function `ProDrawingView2DSectionTotalSet()` sets the visibility of model edges to **Total**. A total cross section shows not only the cross-sectioned area, but also the edges of the model that become visible when a cross section is made. Here model edges behind the section planes as well as section edges are displayed.

The function `ProDrawingView2DSectionAreaSet()` sets the visibility of model edges to **Area**. An area cross section displays only the cross section without the geometry, that is, only section edges are displayed.

View States

Functions Introduced:

- **ProDrawingViewSimplifiedGet()**
- **ProDrawingViewSimplifiedSet()**
- **ProDrawingViewExplodedGet()**
- **ProDrawingViewExplodedSet()**

The functions `ProDrawingViewSimplifiedGet()` and `ProDrawingViewSimplifiedSet()` retrieve and set the simplified representation for a specified drawing view.

The functions `ProDrawingViewExplodedGet()` and `ProDrawingViewExplodedSet()` retrieve and set the exploded state for a specified drawing view.

Example 7: Creating Drawing Views and Accessing their Properties

The sample code in the file `UgNotesColor.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_dwg` creates both general and projection types of drawing views and accesses the view scale, orientation, exploded state, and visible areas of the created drawing views.

Drawing Models

Functions Introduced:

- **ProDrawingSolidsCollect()**
- **ProDrawingSolidsVisit()**
- **ProDrawingSolidAdd()**
- **ProDrawingSolidDelete()**
- **ProDrawingSimprepsCollect()**
- **ProDrawingAsmsimprepAdd()**
- **ProDrawingAsmsimprepDelete()**
- **ProDrawingSolidReplace()**
- **ProDrawingCurrentsolidGet()**
- **ProDrawingCurrentsolidSet()**

The function `ProDrawingSolidsCollect()` outputs an array of the solids attached to the drawing, including those not currently displayed in a view. Function `ProDrawingSolidsVisit()` is a visit function of the usual form which visits the same solids.

The function `ProDrawingSolidAdd()` adds a new solid to a drawing, but does not display it. (To create a drawing view, refer to the Creating Views on page 1243 section.)

`ProDrawingSolidDelete()` deletes a solid from a drawing, provided that solid is not currently displayed in a view.

Functions `ProDrawingSimprepsCollect()`,`ProDrawingAsmsimprepAdd()`, and `ProDrawingAsmsimprepDelete()` are the equivalents to the above functions but take a handle to a simplified rep.

The function `ProDrawingSolidReplace()` replaces a drawing model solid with another solid. The old and new solids must be members of the same family table. The following example code describes this function.

The functions `ProDrawingCurrentsolidGet()` and `ProDrawingCurrentsolidSet()` provide access to the current solid model for a given drawing.

Example 8: Replace Drawing Model Solid with a Solid

The sample code in the file `UgDrawingSolidReplace.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing` shows how to replace a drawing model solid with a solid.

Detail Items

The functions described in this section operate on detail items. Detail items are those drawing items that you create in Creo Parametric in the drawings.

In Creo Parametric TOOLKIT, you have the ability to create, delete, and modify detail items, control their display, and examine what detail items are present in the drawing.

There types of detail items available in Creo Parametric TOOLKIT are:

- Draft entities—Contain the graphical items created in Creo Parametric using the options under the **Sketch** tab, in the **Sketching** group. Some of the items are as follows:
  - Arc
  - Ellipse
  - Line
  - Point
  - Polygon
  - Spline
- Notes—Textual annotations created in Creo Parametric using the command **Annotate ► Note**. They can also contain special symbols.
- Symbol definitions—Named groups of other detail items that the Creo Parametric user can save to disk. You create them in Creo Parametric using the options in the **Annotate ► Symbol** command.
- Symbol instances—Instances of a symbol.
- Draft groups—Groups of detail items such as draft entities, notes, symbol instances, and drawing dimensions. You create them in Creo Parametric using the command **Sketch ► Draft Group**.
- OLE objects—Object Linking and Embedding (OLE) objects embedded in the Creo Parametric drawing file from the **Insert Object** dialog box that opens when you click **Layout ► Object**.

All detail items are identified by DHandles which are equivalent to `ProModelitem`, and inherit from `ProModelitem`. This implies that functions such as `ProSelectionModelitemGet()`, `ProSelectionAlloc()`, and `ProModelitemInit()`, can be used for detail items. The values of the type field for the types of detail item are:

- `PRO_DRAFT_ENTITY`—This type is used for draft entities and OLE objects. Special functions exist to distinguish OLE objects from other detail entities.
- `PRO_NOTE`
- `PRO_SYMBOL_DEFINITION`

- PRO_SYMBOL_INSTANCE
- PRO_DRAFT_GROUP

There is generic detail object called ProDtlitem, whose type field can take any of these values, and is used for arguments to functions that can represent any detail item. The following object handles are used in the more specific cases:

- ProDtlentity
- ProDtlnote
- ProDtlsymdef
- ProDtlsyminst
- ProDtlgroup

Listing Detail Items

Functions Introduced:

- **ProDrawingDtlentitiesCollect()**
- **ProDrawingDtlentityVisit()**
- **ProDrawingDtlnotesCollect()**
- **ProDrawingDtlnoteVisit()**
- **ProDrawingDtlsymdefsCollect()**
- **ProDrawingDtlsymdefVisit()**
- **ProDrawingDtlsyminstsCollect()**
- **ProDrawingDtlsyminstVisit()**
- **ProDrawingDtlgroupsCollect()**
- **ProDrawingDtlgroupVisit()**
- **ProDrawingOLEobjectsVisit()**

The function ProDrawingDtlentitiesCollect() collects all the detail entities in the specified drawing and sheet. Set the input argument *symbol* to NULL if you are collecting a detail item in the drawing. If you are collecting a draft entity in a symbol definition, set *symbol* to specify the owning symbol definition.

Note

The function ProDrawingDtlentitiesCollect() will not collect entities with special symbol definition, such as, datum targets or parametric connector symbols.

The function `ProDrawingDtlentityVisit()` visits all the draft entities stored in the specified drawing and sheet.

Note

The function `ProDrawingDtlentityVisit()` will not visit entities with special symbol definition, such as, datum targets or parametric connector symbols.

The function `ProDrawingDtlnotesCollect()` collects all the notes in the specified drawing. Set the input argument *symbol* to `NULL` if you are collecting a note from the drawing. If you are collecting a note from a symbol definition, set *symbol* to specify the owning symbol definition.

The function `ProDrawingDtlnoteVisit()` visits the notes in the specified drawing.

The function `ProDrawingDtlsymdefsCollect()` collects the symbol definitions in the specified drawing.

The function `ProDrawingDtlsymdefVisit()` visits the symbol definitions in the drawing.

The function `ProDrawingDtlsyminstsCollect()` collects symbol instances in the specified drawing.

The function `ProDrawingDtlsyminstVisit()` visits symbol instances in the specified drawing.

The function `ProDrawingDtlgroupsCollect()` collects groups in the specified drawing.

The function `ProDrawingDtlgroupVisit()` visits groups in the specified drawing.

The function `ProDrawingOLEobjectsVisit()` visits the OLE objects embedded in the model. Specify the visit action and visit filter functions.

Displaying Detail Items

Functions Introduced:

- **ProDtlentityDraw()**
- **ProDtlentityErase()**
- **ProDtlnoteDraw()**
- **ProDtlnoteErase()**
- **ProDtlnoteShow()**

- **ProDtlnoteRemove()**
- **ProDtlsyminstDraw()**
- **ProDtlsyminstErase()**
- **ProDtlsyminstShow()**
- **ProDtlsyminstRemove()**
- **ProDtlgroupDraw()**
- **ProDtlgroupErase()**

Each of the displayable item types has four display functions.

The Show function displays the detail item, such that it is repainted on the next draft regeneration.

The Remove function undraws the detail item permanently, so that it is not redrawn on the next draft regeneration.

The Draw function draws the detail item temporarily, so that it is removed on the next draft regeneration.

The Erase function undraws the detail item temporarily, so that it is redrawn on the next draft regeneration, if it was previously “shown”.

Use the Show function after creating an item, and the Remove function before deleting it. Use the Erase function before modifying an item, and the Draw function afterwards.

Note

These functions require that the drawing must be the currently displayed model. To create or modify detail items in a model that is not currently displayed, use the attributes in the data structures related to Display. For example use `ProDtlnotedataDisplayedSet()` to set the item to be saved with the displayed status turned on, so that the next retrieval of the model will display the item.

Creating, Modifying and Reading Detail Items

Functions Introduced:

- **ProDtlentityCreate()**
- **ProDtlentityDataGet()**
- **ProDtlentityDelete()**
- **ProDtlentityModify()**

- **ProDtlentityErase()**
- **ProDtlentityIsOLEObject()**
- **ProDtlnoteCreate()**
- **ProDtlnoteDataGet()**
- **ProDtlnoteDelete()**
- **ProDtlnoteLineEnvelopeGet()**
- **ProDtlnoteModify()**
- **ProDtlsymdefCreate()**
- **ProDtlsymdefDataGet()**
- **ProDtlsymdefDelete()**
- **ProDtlsymdefModify()**
- **ProDtlsymdefToModelCopy()**
- **ProDtlsyminstCreate()**
- **ProDtlsyminstDataGet()**
- **ProDtlsyminstDelete()**
- **ProDtlsyminstModify()**
- **ProDtlgroupCreate()**
- **ProDtlgroupDataGet()**
- **ProDtlgroupDelete()**
- **ProDtlgroupModify()**

For each of the five detail item types there is an opaque data structure which describes the contents of the detail item. You build the appropriate data structure first, using functions provided for that purpose, and then pass it as input to the appropriate `Create()` function. The `*DataGet()` functions output a filled structure describing an existing detail item. The data structures are built and unpacked by their own functions for that purpose described in the following sections.

The function ProDtlentityDataGet() returns a structure that contains information about the specified detail item, in a drawing or in a symbol definition.

Note

The functions ProDtlentityDataGet(), ProDtlentityCreate(), ProDtlentityDelete(), and ProDtlentityModify cannot access symbol definitions for special symbols, such as, datum targets or parametric connector symbols. For such symbols, the functions return the error PRO_TK_GENERAL_ERROR.

The functions ProDtlnoteDataGet() and ProDtlsyminstDataGet() have an argument for the display mode. Both notes and symbols may contain parameterized text, and the display mode specifies whether the data structure output by the *DataGet() function must contain the text before substitution of the parameters (SYMBOLIC mode), or after the displayed text (NUMERIC mode). If using ProDtlnoteDataGet() as a first step in note modification, always set *mode* to SYMBOLIC or the modification removes the parameterization. Refer to section Detail Note Line Data on page 1267 for more information.

Some data structures contain arrays of, or pointers to, deeper structures which have their own manipulation functions, also described in later sections. Lower level data structures should be built before the upper level ones when creating detail items. The data structures and their member structures are listed below.

- ProDtlentitydata—A draft entity
- ProCurvedata—The 2D geometry of the entity (described in the Core: 3D Geometry on page 170 chapter)
- ProDtlnotedata—A detail note
- ProDtlnoteline—A line of text in a note
- ProDtlnotetext—A segment of text in a line that may have it's own cosmetic properties, such as font, height, and so on
- ProDtlattach—One structure for the attachment of the note itself, and one per leader on the note
- ProDtlsymdefdata—A symbol definition
- ProDtlsymdefattach—The types of attachment support for an instance of this symbol
- ProDtlsyminstdata—A symbol instance
- ProDtlvartext—A variable text substitution

The sequence of calls to create a draft entity containing, for example, a line would be:

1. `ProDtlentitydataAlloc()`—Allocate the entity data (see the section on Draft Entity Data on page 1262).
2. `ProCurvedataAlloc()`—Allocate memory for a curve structure
3. `ProLinedataInit()`—Set the curve structure to describe the required line by initializing a line data structure.
4. `ProDtlentitydataCurveSet()`—Add the curve data to the entity data (see the section on Draft Entity Data on page 1262).
5. `ProDtlentityCreate()`—Create the entity (see the section on Creating, Modifying and Reading Detail Items on page 1258).

> **Note**
>
> You must set the drawing view before attempting to create a detail entity, unless you are creating entities in a symbol definition. The view can be a traditional drawing view obtained through `ProDrawingViewsCollect()`, or the drawing sheet background view obtained from `ProDrawingBackgroundViewGet()`.

The other detail items follow the same principles, although for symbol definitions there are added complexities; these are explained in section Creating a Symbol Definition on page 1278.

The function `ProDtlentityIsOLEObject()` identifies if the specified detail entity is actually an OLE object. For more information on OLE objects refer to Accessing OLE Objects on page 1264.

Each entity item has its own `Delete()` function which removes it permanently from the Creo Parametric drawing.

Each entity item also has its own `Modify()` function which passes a new Data structure.

Use the function `ProDtlnoteErase()` to temporarily undraw the note (see the section on Displaying Detail Items on page 1257).

The function `ProDtlnoteDataGet()` returns the data for the note.

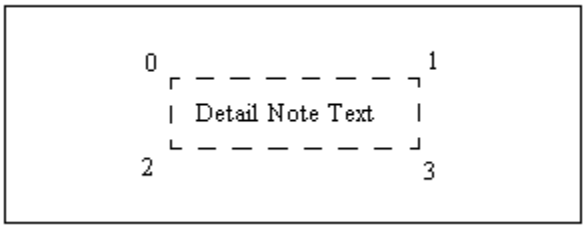
The function `ProDtlnotedataColorSet()` modifies the color in the data (see the section on Detail Note Data on page 1271).

The function `ProDtlnoteModify()` uses the modified data to modify the note itself.

The function `ProDtlnoteDraw()` redraws the note (see the section on Displaying Detail Items on page 1257).

The function `ProDtlnoteLineEnvelopeGet()` determines the screen coordinates of the envelope around a detail note. This envelope is defined by four points. See figure Detail Note Envelope Point Order for how point order is determined.

Detail Note Envelope Point Order



The function `ProDtlsymdefToModelCopy()` copies a specified symbol definition from one model to another.

The function `ProDtlsymdefDelete()` deletes a symbol definition. This function returns the error `PRO_TK_GENERAL_ERROR` when the deletion of a symbol definition from a part fails.

The function `ProDtlsymdefModify()` modifies a symbol definition. The input arguments are listed below:

- *symdef*—Specifies the symbol definition.
- *data*—Specifies the symbol definition data.

Draft Entity Data

Functions Introduced:

- **ProDtlentitydataAlloc()**
- **ProDtlentitydataFree()**
- **ProDtlentitydataIdGet()**
- **ProDtlentitydataCurveGet()**
- **ProDtlentitydataCurveSet()**
- **ProDtlentitydataColorGet()**
- **ProDtlentitydataColorSet()**
- **ProDtlentitydataFontGet()**
- **ProDtlentitydataFontSet()**
- **ProDtlentitydataWidthGet()**
- **ProDtlentitydataWidthSet()**

- **ProDtlentitydataViewGet()**
- **ProDtlentitydataViewSet()**
- **ProDtlentitydataIsConstruction()**
- **ProDtlentitydataConstructionSet()**
- **ProDtlentitydataIsPeriodic()**
- **ProDtlentitydataPeriodicSet()**
- **ProDrawingDraftToDraftent()**

The opaque data structure which describes the contents of a draft entity is called `ProDtlentitydata`.

The only lower-level opaque data structure contained by `ProEntitydata` is `ProCurvedata` which is also used for other 2D and 3D geometry, especially Import Features, and is described elsewhere.

The functions `ProDtlentitydataAlloc()` and `ProDtlentitydataFree()` allocate and free an opaque entity data structure.

Functions `ProDtlentitydataCurveGet()` and `ProDtlentitydataCurveSet()` get and set the geometry of the entity in the form of a `ProCurvdata` object.

`ProDtlentitydataColorGet()` and `ProDtlentitydataColorSet()` get and set the color of the draft entity. The visible data structure `ProColor`, declared in `ProDtlitem.h`, can specify the color in three ways:

- By color index, that is, by choosing one of colors predefined in Creo Parametric, represented by the values of `ProColortype` in `ProToolkit.h`
- By choosing the default color for this type of detail item. (For entities, the default is `PRO_COLOR_DRAWING` and for notes the default is `PRO_COLOR_LETTER`.)
- By specifying the three RGB color values.

If do you do not call `ProDtlentitydataColorSet()` when creating a new entity, the color will be set to be the default color for draft entities.

`ProDtlentitydataFontGet()` and `ProDtlentitydataFontSet()` get and set the line style font which determines the line style used to display the entity. The values are those which appear in the **Line Font** selector in the **Modify Line Style** dialog in Creo Parametric. If you do not call `ProDtlentitydataFontSet()` when creating an entity, the font will be `SOLIDFONT`.

ProDtlentitydataWidthGet() and ProDtlentitydataWidthSet() get and set the line width of the draft entity. The value -1.0 indicates that the entity should have the default width for entities currently set for the drawing. If you do not call ProDtlentitydataWidthSet() when creating a new entity, the width is -1.0.

ProDtlentitydataViewGet() and ProDtlentitydataViewSet() get and set the drawing view to which the entity will be attached. If an entity is attached to a view, it moves whenever the Creo Parametric user moves that view. Entities not attached to a model view must be assigned to the drawing sheet background view instead.

ProDtlentitydataIsConstruction() and ProDtlentitydataConstructionSet() get and set the flag that controls whether the entity is created normal or as a construction entity.

The function ProDtlentitydataIsPeriodic() checks if the draft identity is marked as periodic. The output argument is_periodic is a Boolean. The value PRO_B_TRUE indicates that the draft entity is periodic.

The function ProDtlentitydataPeriodicSet() marks the draft entity to be periodic. The input arguments are as follows:

- *data*—The draft entity data.
- *periodic*—Specify the value PRO_B_TRUE if the draft entity is to be periodic.

Use the function ProDrawingDraftToDraftent() to convert a selection of type draft to draft entity in the specified drawing. The input arguments follow:

- *p_draw*—Specifies the drawing that owns the draft entity.
- *p_sel_draft*—A ProSelection object that represents the selection of type as draft.

The output argument r_p_sel_draft_ent returns the handle to the converted selection using the ProSelection object. The converted selection is managed by the function that calls the function ProDrawingDraftToDraftent().

Example 9: Create a Draft Line with Predefined Color

The sample code in the file UgDtlentityExamples.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing shows a utility that creates a draft line in one of the colors predefined in Creo Parametric.

Accessing OLE Objects

An object linking and embedding (OLE) object is an external file, such as a document, graphics file, or video file that is created using an external application and which can be inserted into another application, such as Creo Parametric. You

can create and insert supported OLE objects into a two-dimensional Creo Parametric file, such as a drawing, report, format file, Notebook, or diagram. The functions described in this section enable you to identify and access OLE objects embedded in drawings.

Function Description

- **ProDrawingOLEobjectSheetGet()**
- **ProDrawingOLEobjectOutlineGet()**
- **ProDrawingOLEobjectApplicationtypeGet()**
- **ProDrawingOLEobjectPathGet()**

The function `ProDrawingOLEobjectSheetGet()` returns the sheet number for the OLE object.

The function `ProDrawingOLEobjectOutlineGet()` returns the extent of the OLE object embedded in the drawing.

The function `ProDrawingOLEobjectApplicationtypeGet()` returns the type of the OLE object as a string, for example, "`Microsoft Word Document`".

The function `ProDrawingOLEobjectPathGet()` returns the path to the external file for each OLE object, if it is linked to an external file.

Detail Note Text Data

Functions Introduced:

- **ProDtlnotetextAlloc()**
- **ProDtlnotetextFree()**
- **ProTextStyleHeightGet()**
- **ProTextStyleHeightSet()**
- **ProTextStyleWidthGet()**
- **ProTextStyleWidthSet()**
- **ProTextStyleSlantAngleGet()**
- **ProTextStyleSlantAngleSet()**
- **ProTextStyleThicknessGet()**
- **ProTextStyleThicknessSet()**
- **ProTextStyleFontGet()**
- **ProTextStyleFontSet()**
- **ProDtlnotetextUlineGet()**
- **ProDtlnotetextUlineSet()**

- **ProDtlnotetextStringGet()**
- **ProDtlnotetextStringSet()**
- **ProDtlnotetextStyleGet()**
- **ProDtlnotetextStyleSet()**
- **ProDtlnoteWrapTextGet()**
- **ProDtlnoteWrapTextSet**

Each line of a drawing note may contain text in several different fonts, heights, and so on. So each line is described in terms of an array of text items, whose contents are described by the data structure `ProDtlnotetext`.

`ProDtlnotetextAlloc()` and `ProDtlnotetextFree()` allocate and free a `ProDtlnotetext` data structure.

`ProTextStyleHeightGet()` and `ProTextStyleHeightSet()` get and set the height of the text. The value -1.0 means that the text has the default height for text currently specified for the drawing.

`ProTextStyleWidthGet()` and `ProTextStyleWidthSet()` get and set the width factor of the text. The width factor is the ratio of the width of each character to the height. The value -1.0 means that the width factor has the default value for text currently specified for the drawing.

`ProTextStyleSlantAngleGet()` and `ProTextStyleSlantAngleSet()` get and set the slant angle of the text.

`ProTextStyleThicknessGet()` and `ProTextStyleThicknessSet()` get and set the line thickness of the text. The value -1.0 means that the text has the default thickness for text currently specified for the drawing.

`ProTextStyleFontGet()` and `ProTextStyleFontSet()` get and set the font used to display the text.

`ProDtlnotetextUlineGet()` and `ProDtlnotetextUlineSet()` get and set whether the text item is underlined. The default is no underline.

`ProDtlnotetextStringGet()` and `ProDtlnotetextStringSet()` get and set the string of characters contained in the text item.

The functions `ProDtlnotetextStyleGet()` and `ProDtlnotetextStyleSet()` retrieve and set the text style for the specified text as a `ProTextStyle` structure. It takes as input the `ProDtlnotetext` object.

`ProDtlnoteWrapTextGet()` and `ProDtlnoteWrapTextSet()` get and set the wrap status of the text for a specified note in a drawing.

The function `ProDtlnoteWrapTextSet()` sets the text wrapping status to ON or OFF. The input arguments are listed below:

- *note*—Specifies the note for which the wrap status is to be set.
- *wrap*—Specifies if the text is wrapped. To wrap the text, specify the value as `Pro_B_True`.
- *wrapwidth*—Specifies the width of the wrapped text line, if the input argument *wrap* is set to `Pro_B_True`.

Detail Note Line Data

Functions Introduced:

- **ProDtlnotelineAlloc()**
- **ProDtlnotelineFree()**
- **ProDtlnotelineTextAdd()**
- **ProDtlnotelineTextsSet()**
- **ProDtlnotelineTextsCollect()**

The `ProDtlnoteline` data structure describes the contents of a single line of text in a detail note.

`ProDtlnotelineAlloc()` and `ProDtlnotelineFree()` allocate and free a `ProDtlnoteline` structure.

`ProDtlnotelineTextAdd()` adds a text item, described by a `ProDtlnotetext` data structure, to a note line. If the line already contained text items, the new one is added at the end of the array.

`ProDtlnotelineTextsSet()` sets the contents of a whole text line, by providing a new array of `ProDtlnotetext` items. If the note line already contained text items, they are replace by the new ones.

`ProDtlnotelineTextsCollect()` outputs an array of the text items contained in a specified text line.

Points to note about Text Lines and parameterization:

- If the string in a Text Line you put in a note contains one or more parameters, Creo Parametric will divide the Text Line into several Text Items to ensure that each parameter has its own Text Item.
- When you look at the text in an existing note by using the function `ProDtlnoteDataGet()` with the *mode* option set to `SYMBOLIC` (that is, to see the text before substitution of the parameters), you will see the text bracketing and text item identifiers that you also see when you edit a text line in Creo Parametric.

  For example, if you make a text line containing a single text item with the text

```
"model = &model_name"
```

Creo Parametric will put the &model_name into a separate text item when the note is created. If you then use `ProDtlnoteDataGet()` on the created note with the *mode* option set to `SYMBOLIC`, you will see the following two text items in the relevant text line

```
"model = "      "&model_name"
```

If you set *mode* to `NUMERIC`, you see these text items:

```
"model = "      "MODEL"
```

where `MODEL` is the name of the model.

Note

Creo Parametric does not resolve and replace symbolic callouts for notes which are not displayed. Therefore, if the note is not displayed or is hidden in a layer, the text retrieved may contain symbolic callouts, even when the *mode* is set to `NUMERIC`.

Note that `ProDtlnotetextStringGet()` does not return the brackets and numbers for each individual text entity. In addition, the function does not return the special escape characters (such as `\}`) to represent characters previously provided.

Refer to the section Creating a Symbol Definition on page 1278 for a description of how to find which Creo Parametric model owns the parameter referred to by parameterized text.

Detail Attachments and Leaders

Functions Introduced:

- **ProDtlattachAlloc()**
- **ProDtlattachGet()**
- **ProDtlattachSet()**
- **ProDtlattachFree()**
- **ProDtlattachArrowtypeGet()**
- **ProDtlattachArrowtypeSet()**

The opaque data structure `ProDtlattach` is used for two tasks:

- The way in which a drawing note or a symbol instance is attached to the drawing.
- The way in which a leader on a drawing note or symbol instance is attached.

Each note and symbol instance must contain one ProDtlattach to describe its attachment in the drawing, and may contain any number of ProDtlattach objects describing the leaders.

ProDtlattachAlloc() allocates and initializes the memory for a detail attachment. The inputs are:

- *type*—The type of attachment to the drawing view. The detail attachment types are as follows:
  - FREE—The attachment is to a 2D location in the drawing view.
  - PARAMETRIC—The attachment is to a point on a surface or an edge of a solid.
  - OFFSET—The attachment is offset to another drawing view, to a model item, or to a 3D model annotation.

> **Note**
>
> You cannot attach a symbol to 3D model annotation using the OFFSET attachment type.

- *view*—The drawing view. If the type is FREE, the attachment is relative to the drawing view, that is the attachment moves when the drawing view is moved. This is NULL, if the detail attachment is not related to the drawing view, but is placed at a specified location in the drawing sheet, or if the attachment is offset to a model item or to a 3D model annotation.
- *location*—If the type is FREE or OFFSET, this argument provides the location of the attachment. This location is in screen coordinates for drawing items, symbol instances and surface finishes on flat-to-screen annotation planes, and in model coordinates for symbols and surface finishes on 3D model annotation planes. The distance from this location to the location of the item to which the detail item is attached (given by the argument *attach_point*) is saved as the offset distance for an OFFSET attachment.
- *attach_point*—If the type is PARAMETRIC or OFFSET, this ProSelection structure provides the location of the item to which the detail item is attached. This includes the drawing view in which the attachment is made. If you are building this structure using ProSelectionAlloc(), set the location using ProSelectionUvParamSet(), and the drawing view using ProSelectionViewSet().

Use the function ProDtlattachSet() to set the above ProDtlattach information for an existing attachment.

The function ProDtlattachGet() unpacks the above information for an existing attachment. The output arguments are:

- *type*—The type of attachment to the drawing view. The detail attachment types are as follows:
  - FREE—The attachment is to a 2D location in the drawing view.
  - PARAMETRIC—The attachment is to a point on a surface or an edge of a solid in a drawing view.
  - OFFSET—The attachment is offset to another drawing view, to a model item, or to a 3D model annotation.
  - UNIMPLEMENTED—The attachment is to an item that is not currently supported in Creo Parametric TOOLKIT. However, you can still retrieve the location and the view to which the attachment is connected.
  - SUPPRESSED—The attachment is to an item, which is missing from the drawing or part.
- *view*—If the type is FREE or UNIMPLEMENTED, this argument specifies the drawing view. This is NULL, if the detail attachment is not related to the drawing view, but is placed at a specified location in the drawing sheet, or if the attachment is offset to a model item or to a 3D model annotation.
- *location*—If the type is FREE, OFFSET, or UNIMPLEMENTED, this argument specifies the location of the attachment. This location is in screen coordinates for drawing items, symbol instances and surface finishes on flat-to-screen annotation planes, and in model coordinates for symbols and surface finishes on 3D model annotation planes. The distance from this location to the location of the item to which the detail item is attached (given by the argument *attach_point*) is saved as the offset distance for an OFFSET attachment.
- *attach_point* —If the type is PARAMETRIC or OFFSET, this argument provides the location of the item to which the detail item is attached. This includes the drawing view in which the attachment is made.

ProDtlattachFree() frees an attachment that was allocated with ProDtlattachAlloc().

The function ProDtlattachArrowtypeGet() returns the type of arrowhead used for the leaders attached to a drawing note or symbol instance. Use the function ProDtlattachArrowtypeSet() to assign the type of arrowhead.

Note

The functions `ProDtlattachArrowtypeGet()` and `ProDtlattachArrowtypeSet()` are applicable only for `ProDtlattach` leader attachment objects obtained using the functions `ProDtlsyminstdataLeadersCollect()` and `ProDtlnotedataLeadersCollect()`.

Detail Note Data

Functions Introduced:

- **ProDtlnotedataAlloc()**
- **ProDtlnotedataFree()**
- **ProDtlnotedataIdGet()**
- **ProDtlnotedataLineAdd()**
- **ProDtlnoteldataLinesSet()**
- **ProDtlnotedataLinesCollect()**
- **ProTextStyleMirrorGet()**
- **ProTextStyleMirrorSet()**
- **ProTextStyleColorGetWithDef()**
- **ProTextStyleColorSetWithDef()**
- **ProDtlnotedataAttachmentGet()**
- **ProDtlnotedataAttachmentSet()**
- **ProDtlnotedataLeadersCollect()**
- **ProDtlnotedataLeadersSet()**
- **ProDtlnotedataLeaderAdd()**
- **ProDtlnotedataElbowlengthGet()**
- **ProDtlnotedataElbowlengthSet()**
- **ProTextStyleAngleGet()**
- **ProTextStyleAngleSet()**
- **ProTextStyleJustificationGet()**
- **ProTextStyleJustificationSet()**
- **ProTextStyleVertJustificationGet()**
- **ProTextStyleVertJustificationSet()**

- **ProDtlnotedataIsDisplayed()**
- **ProDtlnotedataDisplayedSet()**
- **ProDtlnoteDtlsyminstsCollect()**
- **ProDtlnotedataTextStyleGet()**
- **ProDtlnotedataTextStyleSet()**
- **ProDtlnoteTableCellGet()**

The object `ProDtlnotedata` is an opaque pointer to a data structure that describes the contents of a drawing note.

`ProDtlnotedataAlloc()` and `ProDtlnotedataFree()` allocate and free memory for the data.

`ProDtlnotedataIdGet()` gives you the integer id of the note in Creo Parametric that the data describes. This will be set if the `ProDtlnotedata` has been acquired using `ProDtlnoteDataGet()`. It is not necessary to set this when creating a note; the function `ProDtlnoteCreate()` will assign an id to the new note.

`ProDtlnotedataLineAdd()` adds a `ProDtlnoteline` object to a `ProDtlnotedata` description. If the note already contains lines of text, the new line will be added at the end.

`ProDtlnoteldataLinesSet()` sets an array of `ProDtlnoteline` objects as the lines in a `ProDtlnotedata` description. If the note already contains text lines, they will be replaced by the new lines.

`ProDtlnotedataLinesCollect()` outputs an array of `ProDtlnoteline` objects describing the lines in a given `ProDtlnotedata` description.

`ProTextStyleMirrorSet()` specifies the option to mirror the note. Specify the input argument *mirror* to true to mirror the note. `ProTextStyleMirrorGet()` returns the mirroring option specified for the note.

`ProTextStyleColorGetWithDef()` and `ProTextStyleColorSetWithDef()` get and set the color for the note. If you do not call `ProTextStyleColorSetWithDef()` when creating a note, the note will have the default color defined by `PRO_COLOR_METHOD_DEFAULT`. Refer to the Draft Entity Data on page 1262 section for a fuller description of the `ProColor` object.

`ProDtlnotedataAttachmentGet()` and `ProDtlnotedataAttachmentSet()` get and set the `ProDtlattach` object which describes the attachment of the note, that is, where and how it is positioned on the drawing.

ProDtlnotedataLeadersCollect() outputs an array of ProDtlattach objects which described the attachment points of the leaders on the note. ProDtlnotedataLeadersSet() adds and array of leaders to a note, replacing existing leaders. ProDtlnotedataLeaderAdd() adds a new leader to the end of the array of current leaders on a note.

ProDtlnotedataElbowlengthGet() and ProDtlnotedataElbowlengthSet() get and set the length of the elbow that connects each leader to the note. If you do not call ProDtlnotedataElbowlengthSet() when creating a note, there will be no elbow.

The functions ProTextStyleAngleGet() and PProTextStyleAngleSet() get and set the angle of rotation of the note. If you do not call ProTextStyleAngleSet() when creating the note, the rotation defaults to 0.0.

The functions ProTextStyleJustificationGet() and ProTextStyleJustificationSet() return and set the horizontal justification for the text style object. The functions ProTextStyleVertJustificationGet() and ProTextStyleVertJustificationSet() return and set the vertical justification for the text style object. Vertical justification applies only to notes in drawing tables and OnItem notes.

ProDtlnotedataIsDisplayed() and ProDtlnotedataDisplayedSet() retrieve and set the flag that controls whether the note is visible or not. If the note is created with this flag set to true, regenerate the drawing using ProDwgDraftRegenerate() to see the displayed note.

ProDtlnoteDtlsyminstsCollect() returns a list of all the symbol instances that are declared in a detail note via the “sym()” callout format. The symbol instances are returned in the order they are encountered in the note text.

The functions ProDtlnotedataTextStyleGet() and ProDtlnotedataTextStyleSet() retrieve and set the text style for the specified note as a ProTextStyle structure. It takes as input the ProDtlnotedata object.

If the note has texts with different styles, the style returned by the function ProDtlnotedataTextStyleGet() will have mixed state for attributes. The attributes are not the same for all texts. In such a case of mixed attribute, if function such as ProTextStyleFontGet() is called, the error PRO_TK_GENERAL_ERROR is returned.

The function `ProDtlnoteTableCellGet()` returns the information on the rows and columns within a table for the specified table note. The information is given by the following output arguments:

- *table*—Specifies the table.
- *p_row*—Specifies the indexed row that starts at 0.
- *p_col*—Specifies the indexed column that starts at 0.

Example 10: Create Drawing Note at Specified Location with Leader to Surface and Surface Name

The sample code in the file `UgDtlnoteExamples.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing`, shows a function which creates a drawing note at a specified location, with a leader attached to a solid surface, and which shows the name of the surface.

Read-Only Notes

Functions Introduced:

- **ProDtlnotedataReadonlySet()**
- **ProDtlnotedataReadonlyGet()**
- **ProDrawingReadonlyselectionAllow()**

You can make an existing drawing note unselectable by Creo Parametric users if you wish to protect it from modification. The functions `ProDtlnotedataReadonlySet()` and `ProDtlnotedataReadonlyGet()` set and get this property on `ProDtlnotedata` objects. Use function `ProDtlnotedataReadonlySet()` in conjunction with `ProDtlnoteDataGet()` and `ProDtlnoteModify()` to change the setting.

The function `ProDrawingReadonlyselectionAllow()` will temporarily allow the selection of read-only notes.

Parameterized Note Text

Function introduced:

- **ProDtlnoteModelrefGet()**

A note in a drawing, or in a symbol definition, can be parameterized. This means that it contains the name of a parameter from a Creo Parametric model, preceded by a '&'. The '&' and the parameter name are replaced by the value of the parameter when the note is displayed, or when the symbol is instantiated.

The parameterizations in different notes and symbols in a single drawing may refer to parameters on different Creo Parametric models, depending upon the history of the drawing. The function `ProDtlnoteModelrefGet()` allows you to find out which model is referred to by a specific parameter.

Cross-referencing Gtols and Drawing Annotations

The functions described in this section provide the drawing object that represents a shown gtol (if the gtol is shown in the drawing), or vice-versa.

Functions Introduced:

- **ProGtolDtlnoteGet()**
- **ProDtlnoteGtolGet()**

The function `ProGtolDtlnoteGet()` returns the detail note that represents a shown geometric tolerance.

Note

This function returns the first detail note that calls out the geometric tolerance. Creo Parametric does not restrict users to showing only a single version of a geometric tolerance callout.

The function `ProDtlnoteGtolGet()` returns the geometric tolerance shown in a detail note, if applicable.

Cross-referencing 3D Notes and Drawing Annotations

The functions described in this section provide access to the drawing object that represents a shown 3D note, (if the 3D note is shown in the drawing), or vice-versa.

Functions Introduced:

- **ProNoteDtlnoteGet()**
- **ProDtlnoteNoteGet()**

The function `ProNoteDtlnoteGet()` returns a detail note that represents a shown model tree.

Note

This function returns the first detail note that calls out the solid model note. Creo Parametric does not restrict users to showing only a single version of a solid model note callout.

The function `ProDtlnoteNoteGet()` returns the solid model note that is displayed as a detail note, if applicable.

Symbol Definition Attachments

Functions Introduced:

- **ProDtlsymdefattachAlloc()**
- **ProDtlsymdefattachGet()**
- **ProDtlsymdefattachFree()**
- **ProDtlsymdefdataAttachAdd()**
- **ProDtlsymdefdataAttachSet()**
- **ProDtlsymdefdataAttachGet()**

A symbol definition has several different ways in which instances of that symbol can be attached to the drawing. In Creo Parametric users set these attachments from the **General** tab on the **Symbol Definition Attributes** dialog. Each attachment type is described in Creo Parametric TOOLKIT by an opaque data structure called `ProDtlsymdefattach`. This is allocated and filled by the function `ProDtlsymdefattachAlloc()`. The types of attachment are:

- `FREE`—The symbol will have no leaders, and will be attached by a specified location.
- `ON_ITEM`—The symbol will be attached to an entity in the drawing.
- `NORM_ITEM`—The symbol will be attached to an entity, and be rotated to be normal to that entity.
- `LEFT_LEADER`—The attachment is by a leader to a point on an entity at the left of the symbol.
- `RIGHT_LEADER`—T he attachment is by a leader to a point on an entity at the right of the symbol.
- `RADIAL_LEADER`—The attachment is by a leader attached to a circular entity in the symbol.

The input arguments to the function are these

- *type*—The type of attachment
- *entity_id*—The id of the entity in the symbol definition which has the attachment point, if the attachment type is *_LEADER.entity_parameter The "t" value of the location on the entity which forms the attachment point, if the attachment type is *_LEADER.
- *position*—The location in the symbol coordinate system which forms the attachment point, if the attachment type is `FREE`, `ON_ITEM`, or `NORM_ITEM`.

Symbol Definition Data

Functions Introduced:

- **ProDtlsymdefdataAlloc()**
- **ProDtlsymdefdataFree()**
- **ProDtlsymdefdataIdGet()**
- **ProDtlsymdefdataHeighttypeGet()**
- **ProDtlsymdefdataHeighttypeSet()**
- **ProDtlsymdefdataTextrefSet()**
- **ProDtlsymdefdataTextrefGet()**
- **ProDtlsymdefdataElbowGet()**
- **ProDtlsymdefdataElbowSet()**
- **ProDtlsymdefdataTextangfixedGet()**
- **ProDtlsymdefdataTextangfixedSet()**
- **ProDtlsymdefdataScaledheightGet()**
- **ProDtlsymdefdataPathSet()**
- **ProDtlsymdefdataPathGet()**
- **ProDtlsymdefdataNameGet()**

The opaque object `ProDtlsymdefdata` describes the contents of a symbol definition. The functions `ProDtlsymdefdataAlloc()` and `ProDtlsymdefdataFree()` allocate and free this data.

`ProDtlsymdefdataIdGet()` gives you the integer id of the symbol definition in Creo Parametric that the data describes. This will be set if the `ProDtlsymdefdata` has been acquired using `ProDtlsymdefDataGet()`. It is not necessary to set this when creating a symbol definition; the function `ProDtlsymdefCreate()` will assign an id to the new note.

`ProDtlsymdefdataHeighttypeGet()` and `ProDtlsymdefdataHeighttypeSet()` get and set the way in which the size of an instance of this symbol definition is set. The three types are:

- FIXED—The symbol instance height is fixed.
- VARIABLE—The symbol instance height may be modified by the Creo Parametric user.
- TEXTRELATED—The symbol instance height is related to the height of a text item in the definition.

If the height type is TEXTRELATED the functions ProDtlsymdefdataTextrefSet() and ProDtlsymdefdataTextrefGet() set and get the text item in the symbol definition which determines the symbol instance height. The reference is by note id, line index, and text item index.

ProDtlsymdefdataElbowGet() and ProDtlsymdefdataElbowSet() get and set the bit flag representing the elbow of the symbol definition.

ProDtlsymdefdataTextangfixedGet() and ProDtlsymdefdataTextangfixedSet() get and set whether the angle of text in the symbol is fixed.

ProDtlsymdefdataScaledheightGet() returns the height of the symbol definition in inches.

ProDtlsymdefdataPathSet() and ProDtlsymdefdataPathGet() set and get the path and file name of the file in which the symbol definition may be saved. This is used to give the symbol its name.

ProDtlsymdefdataNameGet() gets the name of the symbol definition.

Creating a Symbol Definition

The notes and draft entities that are contained by a symbol definition are created using ProDtlentityCreate() and ProDtlnoteCreate(), using the ProDtlsymdef handle as the *symbol* argument. So you need to create the empty symbol definition first, and then add the notes and entities.

If you want to add parametric leader attachments, using ProDtlsymdefdataAttachAdd() and so on, these identify the entities to which the leaders should attach using the object handles output by the calls to ProDtlnoteCreate() and ProDtlentityCreate() that created them. So these attachment types should also be added after the symbol is created.

So the steps in creating a symbol definition are:

- Allocate a description — ProDtlsymdefdataAlloc()
- Add a FREE attachment — ProDtlsymdefattachAlloc(), ProDtlsymdefdataAttachAdd()
- Create the symbol—ProDtlsymdefCreate()

- Add the notes and entities (as for creating notes and entities in the drawing)
- Add any leader attachments—`ProDtlsymdefattachAlloc()`, `ProDtlsymdefdataAttachAdd()`

Example 11: Create Symbol Definition

The sample code in the file `UgDtlsymdefExamples.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing`, shows a function which creates a symbol definition which contains four line entities forming a box, a note at the middle of the box, and a free attachment.

Retrieving a Symbol Definition from Disk

Function introduced:

- **ProDrawingDtlsymdefRetrieve()**
- **ProDrawingSystemDtlsymdefRetrieve()**

Creo Parametric symbols exist in two different areas: the user-defined area and the system symbols area.

The function `ProDrawingDtlsymdefRetrieve()` enables you to retrieve a symbol definition from the user-defined location designated by the configuration option `pro_symbol_dir`. The symbol definition should have been previously saved to a file using Creo Parametric.

The function `ProDrawingSystemDtlsymdefRetrieve()` retrieves a symbol definition from the system directory. The system area contains symbols provided by Creo Parametric with the Detail module (such as the Welding Symbols Library).

Symbol Instance Variable Text

Functions Introduced:

- **ProDtlvartextAlloc()**
- **ProDtlvartextFree()**
- **ProDtlvartextDataGet()**

A symbol instance may replace any text inside a note in the symbol definition that is enclose in back slash characters. The opaque data structure `ProDtlvartext` describes such a substitution. It describes the “prompt” string, that is, the string in the symbol definition which it is replacing, and the “value”, that is, the new text string.

The function `ProDtlvartextAlloc()` allocates and initializes a `ProDtlvartext` object. `ProDtlvartextFree()` frees the memory, and `ProDtlvartextDataGet()` unpacks the information in a `ProDtlvartext`.

Symbol Instance Data

Functions Introduced:

- **ProDtlsyminstdataAlloc()**
- **ProDtlsyminstdataFree()**
- **ProDtlsyminstdataColorSet()**
- **ProDtlsyminstdataColorGet()**
- **ProDtlsyminstdataDefSet()**
- **ProDtlsyminstdataDefGet()**
- **ProDtlsyminstdataAttachtypeSet()**
- **ProDtlsyminstdataAttachtypeGet()**
- **ProDtlsyminstdataDefattachSet()**
- **ProDtlsyminstdataDefattachGet()**
- **ProDtlsyminstdataAttachmentGet()**
- **ProDtlsyminstdataAttachmentSet()**
- **ProDtlsyminstDimattachGet()**
- **ProDtlsyminstdataLeadersCollect()**
- **ProDtlsyminstdataLeadersSet()**
- **ProDtlsyminstdataLeaderAdd()**
- **ProDtlsyminstdataElbowlengthGet()**
- **ProDtlsyminstdataElbowlengthSet()**
- **ProDtlsyminstdataAngleSet()**
- **ProDtlsyminstdataAngleGet()**
- **ProDtlsyminstdataScaledheightSet()**
- **ProDtlsyminstdataScaledheightGet()**
- **ProDtlsymInstnoteDataGet()**
- **ProDtlsymInstentityDataGet()**
- **ProDtlsyminstdataDisplayedSet()**
- **ProDtlsyminstdataIsDisplayed()**
- **ProDtlsyminstdataIsInvisible()**

- **ProDtlsyminstdataVartextAdd()**
- **ProDtlsyminstdataVartextsSet()**
- **ProDtlsyminstdataVartextsCollect()**
- **ProDtlsyminstdataTransformGet()**
- **ProDtlsyminstdataGroupoptionsSet()**
- **ProDtlsyminstEntitiesVisibleGet()**
- **ProDtlsyminstIsDatumTarget()**
- **ProDtlsyminstEnvelopeGet()**
- **ProDtlsyminstReferencesAdd()**
- **ProDtlsyminstReferencesGet()**
- **ProDtlsyminstReferenceDelete()**

`ProDtlsyminstdataAlloc()` and `ProDtlsyminstdataFree()` allocate and free a `ProDtlsyminstdata` description.

`ProDtlsyminstdataDefSet()` and `ProDtlsyminstdataDefGet()` set and get the reference to the symbol definition that this instance instantiates.

`ProDtlsyminstdataAttachtypeSet()` and `ProDtlsyminstdataAttachtypeGet()` set and get the type of attachment being chosen for the symbol instance. The corresponding attachment types much exist in the symbol definition.

If you want to make an attachment to a symbol instance of a type that was not specified in the symbol definition, you can add you own symbol definition attachment to the symbol instances. `ProDtlsyminstdataDefattachSet()` and `ProDtlsyminstdataDefattachGet()` set and get a `ProDtlsymdefattach` object on a symbol instance with this purpose.

`ProDtlsyminstdataAttachmentGet()` and `ProDtlsyminstdataAttachmentSet()` get and set the actual attachment for the symbol instance, that is, where it is positioned on the drawing, in the form of a `ProDtlattach` object. Refer to the section on Detail Attachments and Leaders on page 1268 for more information about this object.

Note

You cannot attach a symbol to 3D model annotation using the `OFFSET` attachment type. While attaching a symbol to 3D model annotation, if you set the attachment type as `OFFSET`, the function `ProDtlsyminstdataAttachmentSet()` returns the error `PRO_TK_INVALID_TYPE`.

The function ProDtlsyminstDimattachGet() returns the dimension to which the specified symbol instance is attached. The function returns the error of type PRO_TK_BAD_CONTEXT when the dimension to which the specified symbol instance is attached is not available. In this case, the model containing the dimension was either deleted or suppressed in the assembly.

ProDtlsyminstdataLeaderAdd() adds a leader to a symbol instance description.

ProDtlsyminstdataLeadersSet() sets an array of leaders in a symbol instance, replacing any existing leaders.

Note

To remove all the leaders from the symbol instance data, pass NULL as the value for the input argument *leaders* and set the attachment type to PROSYMDEFATTACHTYPE_FREE.

ProDtlsyminstdataLeadersCollect() outputs an array of the leaders on a ProDtlsyminstdata description.

ProDtlsyminstdataElbowlengthGet() and ProDtlsyminstdataElbowlengthSet() get and set the length of the elbow that connects each leader to the symbol instance. If you do not call ProDtlnotedataElbowlengthSet() when creating a symbol instance, there will be no elbow.

ProDtlsyminstdataAngleSet() and ProDtlsyminstdataAngleGet() get and set the rotation angle of the symbol, if the symbol definition allows rotation. (See also the function ProDtlsymdefdataTextangfixedSet() in the section on Symbol Definition Data on page 1277.)

ProDtlsyminstdataScaledheightSet() and ProDtlsyminstdataScaledheightGet() assign and return the height of a symbol instance in the units of the owner drawing or model, respectively. This value is consistent with the height value shown for a symbol instance in the **Properties** dialog box in the Creo Parametric user interface.

Note

The scaled height obtained using the above functions is partially based on the properties of the symbol definition assigned using the function ProDtlsyminstdataDefSet(). Changing the symbol definition may change the calculated value for the scaled height.

`ProDtlsymInstnoteDataGet()` and `ProDtlsymInstentityDataGet()` retrieve the data of a note and an entity, respectively, in the symbol instance.

The function `ProDtlsyminstdataIsDisplayed()` checks if the specified instance is not marked as erased. The function `ProDtlsyminstdataDisplayedSet()` sets the flag which controls whether or not the instance is marked as displayed.

The function `ProDtlsyminstdataIsInvisible()` checks if the specified instance is invisible. An invisible symbol instance will not appear in the drawing even if it marked as displayed. For example:

- if the symbol is in a draft group, which is marked as suppressed
- if the symbol is a BOM balloon, and the repeat region cannot find an appropriate model
- if the symbol is a weld symbol, and its feature is suppressed
- if the symbol is a datum target symbol, and its feature is suppressed

`ProDtlsyminstdataVartextAdd()`, `ProDtlsyminstdataVartextsSet()`, `ProDtlsyminstdataVartextsCollect()` manipulate `ProDtllvartext` objects in the symbol instance, which provide for substitution of text in the symbol definition. See section Symbol Instance Variable Text on page 1279 for more information about the `ProDtlvartext` object.

The function `ProDtlsyminstdataTransformGet()` provides a matrix that describes the transformation between symbol definition coordinates and screen coordinates for this instances, that is, it describes the location and orientation of the symbol. The symbol coordinates are specified in inches.

The function `ProDtlsyminstdataGroupoptionsSet()` sets the option for displaying groups in the symbol instance. The possible options are:

- Interactive—prompt the user to select the groups to activate
- All—activate all groups
- None—do not activate any group
- Custom—activate only those groups included in the array of `ProDtlsymgroup` handles passed to this function.

See the section Drawing Symbol Groups on page 1286 to learn more about accessing groups during symbol placement.

The function `ProDtlsyminstEntitiesVisibleGet()` returns the visible entities in the symbol instance data. The input argument *sym_inst* is the symbol instance that displays the symbol added to the drawing.

The function ProDtlsyminstIsDatumTarget() checks if the specified symbol instance is a datum target. This function returns PRO_B_TRUE if the specified symbol instance is a datum target and returns PRO_B_FALSE if it is not.

The function ProDtlsyminstEnvelopeGet() returns the envelope of the symbol. While retrieving coordinates of the symbol in a specified solid, if the symbol is displayed in the solid as well as in the drawing, the drawing must not be active. The input arguments follow:

- *syminst*—Symbol.
- *drawing*—Drawing. The value for this input argument must be passed only if the solid symbol is shown in the drawing. Else, pass it as NULL.
- *path*—If the value of the input argument *drawing* is not NULL, then the path points to a part in an assembly whose drawing is passed here. This part is the owner of the symbol instance.

The output argument envelope is the envelope surrounding the symbol in the model coordinate system. For drawing, the envelope surrounding the symbol is in the screen coordinates.

The function ProDtlsyminstReferencesAdd() adds semantic references to a specified symbol. The input arguments follow:

- *syminst*—Specifies the symbol to which the semantic references are to be added.
- *refs*—Specifies the array of semantic references using the enumerated data type ProAnnotationReference.

Note

When a reference includes more than one collection, the function ProDtlsyminstReferencesAdd() returns the error PRO_TK_MAX_LIMIT_REACHED and no reference is added.

The function ProDtlsyminstReferencesGet() returns a ProArray of additional semantic references for a symbol.

Use the function ProAnnotationreferencearrayFree() to free the ProArray.

The function `ProDtlsyminstReferenceDelete()` deletes the additional semantic references. The input arguments are as follows:

- *syminst*—Symbol from which the additional semantic references are to be deleted.
- *index_ref*—Specifies the index of the references that need to be deleted. Indices start from 0. Get the existing references from `ProDtlsyminstReferencesGet()`.

Cross-referencing Weld Symbols and Drawing Annotations

The functions described in this section provide a drawing object that represents a shown weld symbol (if the weld symbol is shown in the drawing), or the weld feature that owns a shown weld symbol.

Functions Introduced:

- **ProFeatureDtlsyminstGet()**
- **ProDtlsyminstFeatureGet()**

The function `ProFeatureDtlsyminstGet()` returns the detail symbol instance that represents a shown model symbol.

The function `ProDtlsyminstFeatureGet()` returns the weld feature that owns the shown weld symbol.

Example 12: Create Free Instance of Symbol Definition

The sample code in the file `UgDtlsyminstExamples.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing`, shows a function which creates a free instance of a symbol definition.

Detail Group Data

Functions Introduced:

- **ProDtlgroupdataAlloc()**
- **ProDtlgroupdataFree()**
- **ProDtlgroupdataIdGet()**
- **ProDtlgroupdataNameGet()**
- **ProDtlgroupdataIsDisplayed()**
- **ProDtlgroupdataDisplayedSet()**
- **ProDtlgroupdataItemAdd()**

- **ProDtlgroupdataItemsSet()**
- **ProDtlgroupdataItemsCollect()**

`ProDtlgroupdataAlloc()` and `ProDtlgroupdataFree()` allocate and free a detail group structure in the form of a `ProDtlgroup` object. `ProDtlgroupdataAlloc()` also sets the name of the group.

`ProDtlgroupdataIdGet()` returns the internal ID of an existing group. `ProDtlgroupdataNameGet()` gets the name of the group.

`ProDtlgroupdataDisplayedSet()` and `ProDtlgroupdataIsDisplayed()` set and get the flag that controls whether or not the group is visible.

`ProDtlgroupdataItemAdd()` adds an item to the group contents. Items supported in the groups include entities, notes, symbol instances, and draft drawing dimensions.

`ProDtlgroupdataItemsSet()` sets the array of items into a group structure, replacing any existing items that may have been assigned.

`ProDtlgroupdataItemsCollect()` returns an array of the items in the group structure.

Example 13: Create New Group of Items

The sample code in the file `UgDtlgroupExamples.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing`, shows a command which creates a group from a set of selected detail items.

Drawing Symbol Groups

This section describes Creo Parametric TOOLKIT functions that give access to user-defined groups contained in drawing symbols.

User-defined groups in symbol definitions are represented by the following handle in Creo Parametric TOOLKIT:

```
typedef struct pro_dtlvargroup
{
 ProDtlsymdef symbol_def;
 int var_group_id;
} ProDtlsymgroup;
```

The group handle contains the definition handle and an identifier that is unique in the group definition.

Creo Parametric allows a hierarchal relationship between the groups in a symbol definition. Thus, some groups contain groups, or are parents of other groups. To transmit the “level” in which a group resides to Creo Parametric TOOLKIT functions, pass the `ProDtlsymgroup` handle of the parent group. To look at the groups at the top level, pass a `ProDtlsymgroup` handle with an identifier of -1.

Identifying Symbol Groups in an Instance

Function introduced:

- **ProDtlsyminstSymgroupsCollect()**

The function `ProDtlsyminstSymgroupsCollect()` indicates which groups are included in the symbol instance. You can collect the groups based on their status:

- All—retrieve all groups in the definition of the symbol instance
- Active—retrieve only those groups which are actively shown in this symbol instance
- Inactive—retrieve only those groups which are not shown in this symbol instance

Identifying Symbol Groups in a Definition

Functions Introduced:

- **ProDtlsymgroupSubgroupsCollect()**
- **ProDtlsymgroupDataGet()**
- **ProDtlsymgroupdataNameGet()**
- **ProDtlsymgroupdataItemsCollect()**
- **ProDtlsymgroupParentGet()**
- **ProDtlsymgroupLevelIsExclusive()**

The function `ProDtlsymgroupSubgroupsCollect()` returns the names of all subgroups stored in the symbol definition at the indicated level.

Use the function `ProDtlsymgroupDataGet()` to get the data for the group stored in the symbol definition.

The function `ProDtlsymgroupdataNameGet()` returns the name of the group using the symbol group data as input, while the function `ProDtlsymgroupdataItemsCollect()` returns the names of all the group members using symbol data as input. Note that all these group members are entities or notes contained only within the symbol definition.

The function `ProDtlsymgroupParentGet()` returns the parent group to which the current group belongs.

The function `ProDltsymgroupLevelIsExclusive()` indicates if the subgroups stored in the symbol definition at the current level are exclusive or independent. If groups are exclusive, only one of the groups at this level may be active in the model at any time. If groups are independent, any number of groups may be active.

Manipulating Symbol Groups

Functions Introduced:

- **ProDtlsymgroupdataAlloc()**
- **ProDtlsymgroupdataNameSet()**
- **ProDtlsymgroupdataItemsSet()**
- **ProDtlsymgroupdataItemAdd()**
- **ProDtlsymgroupdataFree()**
- **ProDtlsymgroupSubgroupCreate()**
- **ProDtlsymgroupModify()**
- **ProDtlsymgroupDelete()**
- **ProDtlsymgroupLevelExclusiveSet()**

The opaque handle `ProDtlsymgroupdata` contains the information needed to define or redefine a group.

The function `ProDtlsymgroupdataAlloc()` allocates the data structure.

The function `ProDtlsymgroupdataNameSet()` sets the name of the symbol group while the function `ProDtlsymgroupdataItemsSet()` sets the specified items to be contained in the symbol group, provided such items belong to the symbol definition. The items to be included can be detail entities and notes.

The function `ProDtlsymgroupdataItemAdd()` adds a single item to the symbol group, provided such an item belongs to the symbol definition. The item to be added can be a detail entity or a note.

The function `ProDtlsymgroupdataFree()` frees the data structure.

The function `ProDtlsymgroupSubgroupCreate()` creates a new group in the symbol definition at the specified level below the parent group.

The function `ProDtlsymgroupModify()` modifies the symbol group definition.

The function `ProDtlsymgroupDelete()` deletes a group from the symbol definition. This function does not delete the entities contained in the group.

The function `ProDtlsymgroupLevelExclusiveSet()` makes the groups at the indicated level, exclusive or independent, in the symbol definition.

Drawing Edges

The functions described in this section provide access to the display properties such as color, line font, and thickness of model edges in drawing views. Model edges can be regular edges, silhouette edges, or non-analytical silhouette edges.

The opaque handle `ProDrawingEdgeDisplay` provides access to the display properties of model edges.

Note

You can select model edges from detailed views for modification, but no change will be applied. To modify the display of a model edge in a detailed view, you must select the edge in the parent view.

Functions Introduced:

- **ProDrawingEdgeDisplayGet()**
- **ProDrawingEdgeDisplaySet()**
- **ProDrawingedgedisplayFree()**
- **ProDrawingedgedisplayColorGet()**
- **ProDrawingedgedisplayColorSet()**
- **ProDrawingedgedisplayFontGet()**
- **ProDrawingedgedisplayFontSet()**
- **ProDrawingedgedisplayWidthGet()**
- **ProDrawingedgedisplayWidthSet()**
- **ProDrawingedgedisplayIsGlobal()**
- **ProDrawingedgedisplayGlobalSet()**

The function `ProDrawingEdgeDisplayGet()` obtains the display properties of a specified model edge in a drawing view. It allocates the `ProDrawingEdgeDisplay` object for storing the display properties.

The function `ProDrawingEdgeDisplaySet()` assigns the display properties of a specified model edge in a drawing view. After assigning the properties, you must repaint the drawing view to update the display.

The function `ProDrawingedgedisplayFree()` frees the memory allocated for the `ProDrawingEdgeDisplay` object.

The functions `ProDrawingedgedisplayColorGet()` and `ProDrawingedgedisplayColorSet()` obtain and assign the color to be used for the display of a specified model edge.

The functions ProDrawingedgedisplayFontGet() and ProDrawingedgedisplayFontSet() obtain and assign the line font to be used for the display of a specified model edge.

The functions ProDrawingedgedisplayWidthGet() and ProDrawingedgedisplayWidthSet() obtain and assign the width to be used for the display of a specified model edge. You must pass a value less than zero to use the default width.

Note

The width obtained is in screen coordinates. To convert the width value into drawing coordinates, use the sheet transformation matrix obtained using ProDrawingSheetTrfGet().

The function ProDrawingedgedisplayIsGlobal() determines if the model edge display properties such as color, line font, and width have been applied globally to all the drawing views in the drawing sheet.

The function ProDrawingedgedisplayGlobalSet() sets the flag that assigns the model edge display properties such as color, line font, and width globally to all the drawing views in the drawing sheet.

Drawing Tables

A drawing table is identified by the DHandle ProDwgtable which is typedef to and inherited from ProModelitem. The type field in ProDwgtable has the value PRO_DRAW_TABLE.

Selecting Drawing Tables and Cells

Function Introduced:

- **ProSelectionDwgtblcellGet()**

In order to ask the user to select a table cell, use the option *dwg_table* as input to ProSelect(), and then use ProSelectionModelitemGet() to acquire the ProDwgtable handle to the table.

To select a table cell, use the option *table_cell* and call ProSelectionModelitemGet() to get the table handle, and the special function ProSelectionDwgtblcellGet() that returns the IDs of the selected table segment, column and row. The function ProSelectionDwgtblcellGet() returns row and column values starting from 0. To get the actual values of the rows and the columns, add 1 to the result, so that these can be used in other Creo Parametric TOOLKIT functions.

Creating Drawing Tables

Functions Introduced:

- **ProDwgtabledataAlloc()**
- **ProDwgtabledataOriginSet()**
- **ProDwgtabledataSizetypeSet()**
- **ProDwgtabledataColumnsSet()**
- **ProDwgtabledataRowsSet()**
- **ProDrawingTableCreate()**
- **ProDwgtableTextEnter()**
- **ProDwgtableDisplay()**
- **ProDwgtableGrowthdirectionSet()**

The information required to build a table is contained in an opaque data structure, `ProDwgtabledata` that has to be allocated and filled before the table can be created. The function `ProDwgtabledataAlloc()` allocates the data.

`ProDwgtabledataOriginSet()` sets the position of the top left corner of a table in the `ProDwgtabledata` description.

`ProDwgtabledataSizetypeSet()` specifies whether the size of the columns and rows will be in screen coordinates, or as the number of text characters. It is usually more convenient to specify as numbers of characters.

`ProDwgtabledataColumnsSet()` sets the width of the columns as well as the default justifications of text in the columns.

`ProDwgtabledataRowsSet()` sets the height of the rows.

`ProDwgtableTextEnter()`

`ProDrawingTableCreate()` creates the table in the Creo Parametric drawing and optionally displays it. If your program is about to add rows, columns, or text to the table, it is usually better not to draw it immediately. It can be drawn later using `ProDwgtableDisplay()` and this will avoid multiple redrawing.

The growth direction of a drawing table determines how a drawing table will expand in terms of rows and columns when repeat regions are added to the table. Use the function `ProDwgtableGrowthdirectionSet()` to set the growth direction of the table. The growth direction argument, `ProDwgtableGrowthdirType` takes the following values:

- `PRODWGTABLEGROWTHDIR_DOWNRIGHT`
- `PRODWGTABLEGROWTHDIR_DOWNLEFT`
- `PRODWGTABLEGROWTHDIR_UPRIGHT`
- `PRODWGTABLEGROWTHDIR_UPLEFT`

For more information on the growth direction, see the Creo Parametric Help.

Reading Drawing Tables

Functions Introduced:

- **ProDrawingTableVisit()**
- **ProDrawingTablesCollect()**
- **ProDwgtableInfoGet()**
- **ProDwgtableColumnsCount()**
- **ProDwgtableRowsCount()**
- **ProDwgtableColumnSizeGet()**
- **ProDwgtableRowSizeGet()**
- **ProDwgtableCellNoteGet()**
- **ProDwgtableCelltextGet()**
- **ProDwgtableIsFromFormat()**
- **ProDwgtableRetrieve()**
- **ProDwgtableByOriginRetrieve()**
- **ProDwgtableGrowthdirectionGet()**

`ProDrawingTableVisit()` visits all the tables in a specified drawing; it conforms to the usual standard for visit functions.
`ProDrawingTablesCollect()` is an alternative, and returns an array of `ProDwgtable` handles for a drawing.

A table may be divided into several segments, which are numbered sequentially from 0. The function `ProDwgtableInfoGet()` takes a `ProDwgtable` and a segment ID as input, and fills a data structure that describes the properties of the table. If the segment does not exist, it returns `PRO_TK_NOT_EXIST`. The properties of the table are as follows:

```
int rotation;                The number of 90 degree turns
                                clockwise.
double seg_origin[3];        The screen coordinates of the
                               top left corner of the segment.
int nrows;                   The number of rows.
int ncols;                   The number of columns.
double outline[2][3];        The outline of the segment.
double seg_char_height;      The text height used for the
                               segment.
double table_char_height;    The text height used for the
                               drawing.
double char_width;           The character width factor.
```

The functions `ProDwgtableRowsCount()` and `ProDwgtableColumnsCount()` return the number of rows and columns in a table respectively.

The functions `ProDwgtableColumnSizeGet()` and `ProDwgtableRowSizeGet()` give the column width and row height for a specified table column and row respectively.

The text item in each cell of a drawing table is stored as a detail note. If you need to modify the note in some way, for example the style, you can use the `ProDtlnote*()` functions described in the section on Detail Items on page 1255. The function `ProDwgtableCellNoteGet()` returns the handle to the detail note that represents the text in a specified table cell.

The function `ProDwgtableCelltextGet()` places the text of the table into a string array.

The function `ProDwgtableIsFromFormat()` indicates whether a table was added to the table as a result of importing a format.

The function `ProDwgtableRetrieve()` retrieves a drawing table from a properly formatted Creo Parametric table file, and places it in the specified drawing. It allows you to add a table to a drawing without having to specify all the table properties. This function also supports parameter tables exported in the `CSV` or `TXT` format from the **Parameters** dialog box or using `ProParameterTableExport()`. Refer to the Core: Parameters on page 210 chapter for more information on parameters.

The input arguments are:

- `drawing`—Specifies the drawing in which the table must be retrieved.
- `file_name`—Specifies the name of the drawing table. You must not mention the extension.
- `file_path`—Specifies the path to the drawing table file. The path must be specified relative to the working directory.
- `file_version`—Species the version of the drawing table that must be retrieved. The version 0, represents the latest version of the drawing table.
- `position`—Specifies the coordinates of the point on the drawing sheet, where the table must be placed. The upper-left corner of the table is placed at this point on the drawing sheet. You must specify the value in screen coordinates.
- `solid`—Specifies the model from which data must be copied into the drawing table. If this argument is passed as `NULL`, an empty table is created.
- `simp_rep`—Specifies a handle to the simplified representation in a solid, from which data must be copied into the drawing table. If this argument is passed as `NULL`, and the argument `solid` is not `NULL`, then data from the solid model is copied into the drawing table.

The function `ProDwgtableByOriginRetrieve()` retrieves a drawing table from a properly formatted Creo Parametric table file, and places it in the specified drawing. The function is similar to function `ProDwgtableRetrieve()`,

except that it positions the origin of the table at the specified point in the drawing. Tables can be created with different origins by specifying the option **Direction**, in the **Insert Table** dialog box.

The function `ProDwgtableGrowthdirectionGet()` gets the growth direction of the table using the enumerated type `ProDwgtableGrowthdirType`. For more information on the values of `ProDwgtableGrowthdirType`, see the section Creating Drawing Tables on page 1291.

Modifying Drawing Tables

Functions Introduced:

- **ProDwgtableRowAdd()**
- **ProDwgtableColumnAdd()**
- **ProDwgtableRowDelete()**
- **ProDwgtableColumnDelete()**
- **ProDwgtableColumnWidthSet()**
- **ProDwgtableRowHeightSet()**
- **ProDwgtableRowheightAutoadjustGet()**
- **ProDwgtableRowheightAutoadjustSet()**
- **ProDwgtableCellsMerge()**
- **ProDwgtableCellsRemesh()**
- **ProDwgtableCelltextWrap()**
- **ProDwgtableSave()**
- **ProDwgtableRotate()**
- **ProDwgtableErase()**
- **ProDwgtableDelete()**

`ProDwgtableRowAdd()` and `ProDwgtableColumnAdd()` can add a row or a column before or after an existing row or column. The input arguments *height_in_chars* and *width_in_chars* specify the row height and column width in characters, respectively. Another input argument *display* specifies whether the updated table should be displayed. When making many changes to a table, it is advisable not to display them immediately, but to use `ProDwgtableDisplay()` to update the display later.

`ProDwgtableRowDelete()` and `ProDwgtableColumnDelete()` delete any specified row or column, including removing the text from the affected cells.

ProDwgtableColumnWidthSet() and ProDwgtableRowHeightSet() assign the width of a specified column and the height of a specified row, respectively, depending upon the size of the drawing table. The drawing table size given by the enumerated data type ProDwgtableSizetype can be of the following types:

- PRODWGTABLESIZE_CHARACTERS—Specifies the size in characters. If the specified value for width of a column or height of a row is a fraction, PRODWGTABLESIZE_CHARACTERS rounds down the fractional value to the nearest whole number.
- PRODWGTABLESIZE_SCREEN—Specifies the size in screen coordinates.
- PRODWGTABLESIZE_CHARS_TRUE—Specifies the size in characters. It enables you to specify a fractional value for the width of a column and height of a row.

To accommodate a wrapped text in a table row, you can use the Creo Parametric TOOLKIT functions to automatically adjust the height of the row to accommodate the entire text content. The functions ProDwgtableRowheightAutoadjustGet() and ProDwgtableRowheightAutoadjustSet() get and set the automatic row height adjustment property for a row of a drawing table. These functions use the enumerated type ProDwgtableRowheightAutoadjusttype, which has the following values:

- PRODWGTBLROWHEIGHT_AUTOADJUST_FALSE—Specifies that the automatic row height adjustment property is not set.
- PRODWGTBLROWHEIGHT_AUTOADJUST_TRUE—Specifies that the automatic row height adjustment property is set.
- PRODWGTBLROWHEIGHT_AUTOADJUST_TRUE_LEGACY—Specifies a pre-Creo Parametric 1.0 release behavior. In this behavior, sometimes the row height may be automatically adjusting and sometimes may not be automatically adjusting. To set an explicit row adjustment status use the function ProDwgtableRowheightAutoadjustSet().

Note

- The value PRODWGTBLROWHEIGHT_AUTOADJUST_TRUE_LEGACY is not applicable to the function ProDwgtableRowheightAutoadjustSet().
- When using the function ProDwgtableRowheightAutoadjustSet() any changes in the height of a row will be seen only after the next regeneration, or a call to the function ProDrawingTablesUpdate().

ProDwgtableCellsMerge() allows the merging of cells within a specified range of rows and columns, to form a single cell. The new cell can be addressed (for example, when using ProDwgtableTextEnter(), or other calls to ProDwgtableCellsMerge()) by the row and column number of the original top left cell. Rows below, and columns to the right, retain their original numbers. The function ProDwgtableCellsRemesh() unmerges all the merged cells in a specified range of rows and columns.

ProDwgtableCelltextWrap() wraps the text in a cell.

ProDwgtableSave() saves a drawing table in one of the formats listed below. The formats given by the enumerated type ProDwgtableFormattype can be of the following types:

- PRODWGTABLEFORMAT_TBL—Specifies the tabular format.
- PRODWGTABLEFORMAT_TXT—Specifies the text format.
- PRODWGTABLEFORMAT_CSV—Specifies the CSV format.

ProDwgtableRotate() rotates a table 90 degrees clockwise.

ProDwgtableErase() erases a drawing table.

ProDwgtableDelete() deletes a drawing table.

Notification Functions

Creo Parametric TOOLKIT notifications are available when a drawing table or a row from a drawing table is deleted. The notification functions are established in a session using the function ProNotificationSet().

- **ProDwgtableDeletePreAction()**
- **ProDwgtableDeletePostAction()**
- **ProDwgtableRowDeletePreAction()**
- **ProDwgtableRowDeletePostAction()**

The notification function ProDwgtableDeletePreAction() is called before deletion of a drawing table. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_DWGTABLE_DELETE_PRE.

The notification function ProDwgtableDeletePostAction() is called after deletion of a drawing table. This function is available by calling ProNotificationSet() with the value of the notify type as PRO_DWGTABLE_DELETE_POST.

The notification function ProDwgtableRowDeletePreAction() is called before the deletion of a row from the selected drawing table. The input arguments for this function are as follows:

- *table*—The table containing the row to be deleted.
- *i_row*—The index of the row to be deleted.

Note

The index of the first row is 1.

This function is available by calling `ProNotificationSet()` with the value of the notify type as `PRO_DWGTABLE_ROW_DELETE_PRE`. The specified row is not deleted if the application returns an error from this callback. If the user cancels the deletion, an appropriate message should be displayed, if required.

The notification function `ProDwgtableRowDeletePostAction()` is called after the deletion of a row from the selected drawing table. This function is available by calling `ProNotificationSet()` with the value of the notify type as `PRO_DWGTABLE_ROW_DELETE_POST`.

Example 14: Creation of Table Listing Datum Points

The sample code in the file `UgDwgtableExamples.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing`, shows a command that creates a drawing table listing the datum points in a model shown in a drawing view.

Drawing Table Segments

Functions Introduced:

- **ProDwgtableSegMove()**
- **ProDwgtableSegCount()**
- **ProDwgtableSegSheetGet()**
- **ProDwgtableSegmentOriginSet()**

Drawing tables can be constructed with one or more segments. Each segment can be independently placed.

Note

For these functions, pass -1 to refer to the only segment of a one-segment table.

Move a drawing table segment to new screen coordinates with the function `ProDwgtableSegMove()`. Pass the coordinates of the target position in format x, y, z=0 to this the function. It moves the table segment to the target position.

Determine the number of segments in a table with the function `ProDwgtableSegCount()`. Pass the name of the table to the function and it returns the number of table segments.

Use the function `ProDwgtableSegSheetGet()` to determine which sheet contains a specified drawing table segment.

Use the function `ProDwgtableSegmentOriginSet()` to assign the origin for a specified drawing table segment.

Repeat Regions

Functions Introduced:

- **ProDwgtableCellIsComment()**
- **ProDwgtableCellComponentGet()**
- **ProDwgtableCellRefmodelGet()**
- **ProDrawingTablesUpdate()**

The functions `ProDwgtableCellIsComment()`, `ProDwgtableCellComponentGet()`, `ProDwgtableCellRefmodelGet()`, and `ProDrawingTablesUpdate()` apply to repeat regions in drawing tables.

`ProDwgtableCellIsComment()` indicates whether a cell in a repeat region contains a comment.

`ProDwgtableCellComponentGet()` returns the full path to the component referenced in a cell in a repeat region of a drawing table. However, this function does not return a valid path if the cell has the attribute `NO DUPLICATE` or `NO DUPLICATE/LEVEL` since there is no unique path available. In this case, use `ProDwgtableCellRefmodelGet()` to return the reference assembly and component referred to by the cell in a repeat region. This function differs from`ProDwgtableCellComponentGet()` such that it returns reference objects, even if the cell attribute is set to `NO DUPLICATE` or `NO DUPLICATE/LEVEL`.

The function `ProDrawingTablesUpdate()` updates repeat regions in all tables to account for changes to the model or models. It is equivalent to the Creo Parametric command `Table`,`Repeat Region`, or `Update`. The drawing must be displayed in the current window.

Note

You must call the function `ProMdlDisplay()` to display the drawing before using the function `ProDrawingTablesUpdate()`.

Creating BOM Balloons

BOM balloons are circular callouts created in an assembly drawing. They shows the Bill of Materials information for each component. You can add the BOM balloons in the drawing using the functions described in this section.

Before you can add BOM balloons, you must create a table, add the repeat region, enter the desired report symbols, and designate the BOM balloon region. After this you can show BOM balloons on a selected assembly view.

Functions Introduced:

- **ProDwgtableCellRegionGet()**
- **ProBomballoonCreate()**
- **ProBomballoonAllCreate()**
- **ProBomballoonByComponentCreate()**
- **ProBomballoonByRecordCreate()**
- **ProBomballoonClean()**

The function `ProDwgtableCellRegionGet()` returns the ID of the repeat region. You must specify the name of the table, the column and row ID as input parameters.

The function `ProBomballoonCreate()` creates the BOM balloons at the specified view. The input arguments are:

- *pro_drawing*—Specifies the name of the drawing.
- *pro_table*—Specifies the name of the table that contains the repeat region and the bill of material.
- *region_id*—Specifies the ID of the repeat region that contains the bill of material. If the ID of the repeat region in the table is `-1`, use the repeat region with ID as `0` in the table.
- *pro_view*—Specifies the view where the balloons must be added.

The function `ProBomballoonAllCreate()` creates the BOM balloons to the first view of the drawing.

The function `ProBomballoonByComponentCreate()` creates the balloons at the specified view and on the specified component. If the view is specified as `NULL`, the balloons are added to the first view of the drawing. Specify the path to the component as *component_memb_id_tab*.

The function `ProBomballoonByRecordCreate()` creates the balloons for the specified record in the BOM table. The input arguments are:

- *pro_drawing*—Specifies the name of the drawing.
- *pro_table*—Specifies the name of the table that contains the repeat region and the bill of material.
- *region_id*—Specifies the ID of the repeat region that contains the bill of material.If the ID of the repeat region in the table is `-1`, use the repeat region with ID as `0` in the table.
- *pro_view*—Specifies the view where the balloons must be added. Specify this argument to create the balloons without a leader.
- *table_record_index*—Specifies the record in the BOM table. The balloons are created at the first component that matches the specified record.
- *reference_memb_id_tab*—Specifies the path to the component. This path is used as reference for the leader of the balloon.
- *reference_id*—Specifies the ID of the component. When the ID is set to `K_NOT_USED`, the balloons are attached without leaders.
- *reference_type*—Specifies the type of component using the enumerated data type `ProType`.
- *attach_note_location*—Specifies a `ProArray` of the attachment point for the balloons on the component.

The function `ProBomballoonClean()` cleans up the location and display of BOM balloons in the specified view. The input arguments are:

- *pro_drawing*—Specifies the name of the drawing.
- *pro_view*—Specifies the view where BOM balloons have been added.
- *clean_pos*—Specifies a boolean value to indicate if the balloon must be cleaned.
- *existing_snap_lines*—Specifies a boolean value to indicate if the existing snap lines must be used for the clean up.
- *offset_from_view_outline*—Specifies the offset distance for the balloon placement from the view outline.
- *stagger*—Specifies a boolean value to indicate if the balloons must be staggered at different offset distance from the view outline.
- *create_stagger_snap_lines*—Specifies the incremental value for the stagger distance between the snap lines.

- *interballoons_distance*—Specifies the distance between the BOM balloons.
- *attach_to_surface*—Specifies if the leaders of the balloon must point to edges or surfaces. To attach the leader to a surface specify the value as `PRO_B_TRUE`.

Drawing Dimensions

This section describes Creo Parametric TOOLKIT functions that give access to the types of dimension that can be created in drawing mode. They do not apply to dimensions which are created in solid mode, either those created automatically as a result of feature creation, or reference dimensions created in a solid.

The `ProDimension` object is introduced in the section on Dimensions on page 566; read the explanation of `ProDimension` at the start of that section before reading further.

Dimensions created in drawing mode are stored either in the solid or in the drawing, depending upon the setting of the `config.pro` option `CREATE_DRAWING_DIMS_ONLY`. The default is `NO`, meaning that the dimensions will be stored in the solid. Refer to the Creo Parametric Detailed Drawings Help for more information on the various types of created dimension, and their behavior.

The `owner` field in the `ProDimension` object always refers to the model in which the dimension is stored.

The function `ProDrawingDimensionVisit()`, described in the section on Dimensions on page 566, can be used to find all the dimensions stored in a drawing.

Drawing Dimension Attachments and Dimension Creation

Functions Introduced:

- **ProDrawingDimAttachpointsGet()**
- **ProDrawingDimAttachpointsViewGet()**
- **ProDrawingDimensionCreate()**

The function `ProDrawingDimAttachpointsGet()` retrieves the entities to which a dimension is attached and the type of attachments. This is applicable only for dimensions created in the drawing mode.

Note

Dimensions created in solid mode are stored in a different way from those created in a drawing, because of their different role, and their attachments are not accessible to this function. If the function is called for a function created in solid mode, it will return an error.

The information about the entities to which the dimension is attached is given by the following output arguments:

- *attachments_arr*—Specifies a `ProArray` of entities to which a dimension is attached. Each attachment point is described by two consecutive array elements, of which the second one may be `NULL`. If both elements are not `NULL`, then the attachment point refers to the intersection of the elements. If the second element is `NULL`, then the attachment point refers to the first one.
- *dsense_arr*—Specifies a `ProArray` of `ProDimSense` that gives more information about how the dimension attaches to the entities.

`ProDimSense` is declared in header file `ProDimension.h`. This is the declaration:

```
typedef struct pro_dim_sense {
    ProDimSenseType       type;
    int                   sense;
    ProDimAngleSense      angle_sense;
    ProDimOrient          orient_hint;
} ProDimSense;
```

The `type field` indicates what type of information is being provided by the `sense` and/or `angle_sense` fields. The following sections list the values of `ProDimSenseType` with an explanation of the value of `sense` and `angle_sense` needed in each case.

Note

Some of the explanations below refer to the direction of an entity. Each entity (which includes 3D edges and curves, and 2D draft entities) has an inherent direction which is the direction in which its parameter "t" increases. In the data structure description of the entity, `ProCurvedata`, the first end specified is always the end at which t=0. For example, the direction of a line entity is from field `end1` to `end2` in `ProLinedata`.

- `PRO_DIM_SNS_TYP_NONE` — In this case, no other information is needed to describe the attachment points. For example, if there is a single attachment which is a straight line, the dimension is the length of the line; if the attachments are two parallel lines, the dimension is the distance between them.
- `PRO_DIM_SNS_TYP_PNT` — In this case the `sense` field is set to a value of the enum `ProPointType` (declared in the header), which specifies the part of the entity to which the dimension is attached. The possible values are these:
  - `PRO_POINT_TYP_END1` — The first end (that is, where "t" = 0)
  - `PRO_POINT_TYP_END2` — The second end (that is, where "t" = 1.0)
  - `PRO_POINT_TYP_CENTER` — The center, if entity is an arc or a circle
  - `PRO_POINT_TYP_NONE` — This is equivalent to setting `type` to `PRO_DIM_SNS_TYP_NONE`
  - `PRO_POINT_TYP_MIDPT` — The midpoint of the entity (where "t" = 0.5)
- `PRO_DIM_SNS_TYP_SPLN_PNT` — This means that the attachment is to a point of a spline. The `sense` field is set to the index of the spline point.
- `PRO_DIM_SNS_TYP_TGT_IDX` — The dimension attaches to a tangent of the entity, which is an arc or circle. The `sense` field is set to the index of the tangent in a list of all possible tangents ordered by the "t" value at which they touch the entity.

- PRO_DIM_SNS_TYP_LIN_AOC_TGT — The dimension is the perpendicular distance between a line and a tangent to an arc or a circle which is parallel to the line. The value of sense is one of the values of the enum ProDimLinAocTgtSense. If the two possible tangents are on different sides of the line entity (because the distance from the line to the center is less than the radius) then the two tangents are distinguished as left or right of the line (with respect to its natural direction). If the two tangents are on the same side of the line (because the distance from the line to the center is more than the radius), the two tangents are distinguished as on the same side of the arc/circle center (0) or on the opposite side (1). There is an enum value for each of the four possible combinations of ways to identify a tangent, though of course only two are possible for a particular line and arc/circle pair.

The four values of ProDimLinAocTgtSense are:

- PRO_DIM_LIN_AOC_TGT_LEFT0—The tangent is to the left of the line, and on the same side of the center of the arc/circle as the line.
- PRO_DIM_LIN_AOC_TGT_RIGHT0—The tangent is to the right of the line, and on the same side of the center of the arc/circle as the line.
- PRO_DIM_LIN_AOC_TGT_LEFT1—The tangent is to the left of the line, and on the opposite sid of the center of the arc/circle to the line.
- PRO_DIM_LIN_AOC_TGT_RIGHT1—The tangent is to the right of the line, and on the opposite side of the center of the arc/circle to the line.
- PRO_DIM_SNS_TYP_ANGLE — The dimension is the angle between two straight entities. The field angle_sense is given by the structure ProDimAngleSense which contains three boolean fields. They have the following meaning:
  - is_first—Is set to TRUE if the angle dimension starts from this entity in a counterclockwise direction; FALSE if the dimension ends at this entity. The value must be TRUE for one entity, and FALSE for the other.
  - should_flip—If should_flip is FALSE, and the entity's inherent direction is away from the angle vertex, then the dimension attaches directly to the entity. If the entity's direction is towards the angle vertex, the dimension is attached to a witness line which is in line with the entity but on the opposite side of the angle vertex—If should_flip is TRUE, then these cases are interchanged.
  - pic_vec_dir—Reserved for future use.
- PRO_DIM_SNS_TYP_PNT_ANGLE—The dimension is the angle between a line entity and the tangent to a curved entity at one of its ends. The curve attachment is of this type. (The line attachment is of the type PRO_DIM_

SNS_TYP_PNT described above.) In this case both the angle and angle_sense fields must be set: sense shows which end of the curve the dimension is attached to; angle_sense shows the direction in which the dimension rotates and in which side of the tangent it attaches.

The field orient_hint describes the orientation of the dimension in cases where this cannot be deduced from the attachments themselves. (When such a dimension is created interactively in Creo Parametric, the user is prompted for the extra information.) For example, if the attachments are datum points that are not vertically or horizontally aligned, Creo Parametric needs to know whether the dimension is to be horizontal, vertical, or slanted. The hint refers to the dimension itself, not the attachment, although it is a field in ProDimSense. Creo Parametric looks at the value of orient_hint in the first item in the ProDimSense array you provide.

The values of ProDimOrient are:

- PRO_DIM_ORNT_NONE—No orientation information is needed or provided.
- PRO_DIM_ORNT_HORIZ—The dimension is horizontal
- PRO_DIM_ORNT_VERT—The dimension is vertical
- PRO_DIM_ORNT_SLANTED—The dimension is slanted
- PRO_DIM_ORNT_ELPS_RAD1—The major diameter of an ellipse
- PRO_DIM_ORNT_ELPS_RAD2—The minor diameter of an ellipse
- PRO_DIM_ORNT_ARC_ANG—The angle of an arc
- PRO_DIM_ORNT_ARC_LENGTH—The length of an arc
- PRO_DIM_ORNT_LIN_TANCRV_ANG—If the dimension is attached to a line and an end point of a curve, the default dimension will be a linear dimension showing the distance between the line and the curve point. If you want the dimension to show instead the angle between the line and the tangent at the curve point, set “orient_hint” to this value.

The function ProDrawingDimAttachpointsViewGet() retrieves the attachments and sense of the specified drawing dimension. This is applicable only for dimensions that are created in the drawing mode. This function fetches and interprets the attachment in the context of the view in which the dimension is placed. The function ProDrawingDimAttachpointsViewGet() supports drawing dimensions that are created from intersections of geometric entities.

The information about the entities to which the dimension is attached is given by the following output arguments:

- *attachments_arr*—Specifies a ProArray of entities to which a dimension is attached. Each attachment point is described by two consecutive array elements, out of which the second one may be a NULL. If both elements are

not NULL, the attachment point refers to the intersection of the elements. If the second element is NULL, the attachment point refers to the first one.

- *dsense_arr*—Specifies a ProArray of ProDimSense that provides more information on how the dimension attaches to the entities.

The function ProDrawingDimCreate() has been deprecated. Use the function ProDrawingDimensionCreate() instead. The function ProDrawingDimensionCreate() creates a dimension in a drawing. It takes as input an array of ProSelection objects and an array of ProDimSense structures that describe the required attachments. It will store the new dimension in the solid or the drawing depending upon the setting of the config.pro option CREATE_DRAWING_DIMS_ONLY. Specify the orientation of the dimension in the input argument *orient_hint*. You can create dimensions that have intersection type of reference. The intersection type of reference is a reference that is derived from the intersection of two entities. Refer to the Creo Parametric Detailed Drawings Help for more information on intersection type of reference.

The dimension will be added to the drawing view specified in the ProSelection object. If you want to build the attachment ProSelection object programmatically by calling ProSelectionAlloc(), rather than interactively using ProSelect(), call the function ProSelectionViewSet() to ensure that your ProSelection specifies the drawing view.

The function outputs a ProDimension object to identify the new dimension.

Ordinate Dimensions

Functions Introduced:

- **ProDrawingDimIsOrdinate()**
- **ProDrawingOrdbaselineCreate()**
- **ProDrawingDimToOrdinate()**
- **ProDrawingDimToLinear()**

The function ProDrawingDimIsOrdinate() tells you whether a particular dimension is an ordinate dimension. If so, it also outputs a ProDimension object to identify the baseline dimension being referenced.

The function ProDrawingOrdbaselineCreate() converts a specified dimension to an ordinate baseline dimension. The choice of which end of the dimension becomes the baseline is made by an input of type ProVector which should be close to the appropriate attachment entity, and be in 3D solid coordinates. The function outputs a new ProDimension object which is used to identify the baseline dimension when converting to ordinate other dimensions which should share that baseline.

The function `ProDrawingDimToOrdinate()` converts an existing linear dimension to ordinate. It requires as one of its inputs a `ProDimension` object that was output from `ProDrawingOrdbaselineCreate()` as input to identify the baseline.

The function `ProDrawingDimToLinear()` converts an existing ordinate dimension to linear.

The last three functions in this section all update the display of the dimension if it is currently displayed.

Example 15: Command Creation of Datum Point Table

The sample code in the file `UgDrawingDimensions.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_drawing`, shows a command which creates vertical and horizontal ordinate dimensions from each datum point in a model in a drawing view to a selected coordinate system datum.

Other Drawing Dimension Properties

Functions Introduced:

- **ProDrawingDimIsAssociative()**
- **ProDrawingDimensionIsDisplayed()**
- **ProDrawingDimensionIsToleranceDisplayed()**
- **ProDrawingDimensionViewGet()**
- **ProDrawingDimSwitchView()**
- **ProDrawingDimensionPosGet()**
- **ProDrawingDimensionTextstyleGet()**
- **ProDrawingDimensionMove()**
- **ProDrawingDualDimensionGet()**
- **ProDimensionDualOptionsGet()**
- **ProDimensionDualOptionsSet()**
- **ProDrawingDimensionPathGet()**

If you create a dimension which attaches only to draft entities in the drawing, the dimension may be associative or non-associative. This setting depends on the setting of the drawing setup option `associative_dimensioning`. The associative status of a dimension remains even when the setup option changes, so a drawing may contain dimensions of both types. The function `ProDrawingDimIsAssociative()` tells you whether or not a particular dimension is associative. Refer to the *Drawing User's Guide* for a fuller description of associative draft dimensions.

The function `ProDrawingDimensionIsDisplayed()` identifies whether a drawing dimension is displayed in the drawing.

The function `ProDrawingDimensionIsToleranceDisplayed()` identifies whether the tolerance value is displayed on the drawing dimension.

The function `ProDrawingDimensionViewGet()` tells you what drawing view a dimension is being displayed in. `ProDrawingDimSwitchView()` allows you to switch it's display to another view. Note that not all views will support the display of particular dimension.

The function `ProDrawingDimensionPosGet()` returns the position of the center of the text box for the specified dimension. The coordinates returned by this function cannot be used in the function `ProDrawingDimensionMove()`. Use the function `ProDimlocationTextGet()` instead of the function `ProDrawingDimensionPosGet()`.

The function `ProDrawingDimensionTextstyleGet()` retrieves the text style assigned to the specified dimension or reference dimension. From Creo Parametric 2.0 M190 onward, the function `ProDrawingDimensionTextstyleGet()` has been superseded by the function `ProAnnotationTextstyleGet()`.

The function `ProDrawingDimensionMove()` allows you to move the dimension text to a new position on the drawing. Use the function `ProDimlocationTextGet()` to get the position before and after the move.

The function `ProDrawingDualDimensionGet()` identifies whether a drawing is using dual dimensioning, and also specifies the properties of the dual dimensioning.

The function `ProDimensionDualOptionsGet()` gets information about the display options for the specified dual dimension. In the input argument *drawing*, specify the drawing in which the dimension is displayed. To specify a dimension in owner model, specify the argument value as `NULL`. The output arguments are:

- *type*—From Creo Parametric 5.0.0.0 onward, this argument is no longer supported. Specifies the type of display for primary and secondary dimension using the enumerated data type `ProDualDimensionDisplayType`. The valid values are:
  - `PRO_SECONDARY_DIM_DISPLAY_OFF`—Specifies that secondary dimension is not displayed in a dual dimension.
  - `PRO_SECONDARY_DIM_DISPLAY_BOTTOM`—Specifies that the secondary dimension must be placed below the primary dimension.
  - `PRO_SECONDARY_DIM_DISPLAY_RIGHT`—Specifies that the secondary dimension on the same line as the primary dimension, on the right side.
- *secondary_unit*—From Creo Parametric 5.0.0.0 onward, this argument is no longer supported. Specifies the name of the unit for secondary dimension.

- *dim_decimals*—Specifies the number of decimal places for the secondary dimension.
- *tol_decimals*—Specifies the number of decimal places for tolerance in the secondary dimension.

Use the function `ProDimensionDualOptionsSet()` to set the display options for dual dimensions. From Creo Parametric 5.0.0.0 onward, the arguments *type* and *secondary_unit* are no longer supported for the function `ProDimensionDualOptionsSet()`.

The function `ProDrawingDimensionPathGet()` extracts the component path for a dimension displayed in a drawing.

60

Production Applications: Sheetmetal

This chapter describes the sheet metal geometry analysis and bend table functions. It also introduces and describes the feature element trees for the sheet metal features.

Geometry Analysis

Creo Parametric TOOLKIT geometry analysis functions provide for analysis of sheet metal part geometry and ensure effective customization of sheet metal parts. These analyses include extracting part thickness data and obtaining edge and surface data for sheet metal components.

In addition, sheet metal bend edge and bend surface functions support analyses that:

- Extract bend information associated with bend lines (K-factor, Y-factor, bend deduction, bend allowance).
- Find bend lines when a part is in a flat state.
- Map flat state IDs to bent state IDs.

Functions Introduced:

- **ProSmtPartThicknessGet()**
- **ProSmtSurfaceTypeGet()**
- **ProSmtedgeContourGet()**
- **ProSmtOppsurfGet()**
- **ProSmtOppedgeGet()**
- **ProSmtBendsrfParentGet()**
- **ProSmtBendsrfChildGet()**
- **ProSmtBendedgeChildGet()**
- **ProSmtBendedgeParentGet()**
- **ProSmtMdlIsFlatStateInstance()**
- **ProFaminstanceIsFlatState()**
- **ProSmtBendsrfInfoGet()**

The function `ProSmtPartThicknessGet()` returns the dimension that defines the thickness of the specified sheet metal component. If the model contains the thickness parameter, then this dimension cannot be modified directly. Use the function `ProParameterValueWithUnitsSet()` to assign the value of the thickness parameter. If you specify a non sheet metal part, `ProSmtPartThicknessGet()` returns `PRO_TK_BAD_CONTEXT`.

The function `ProSmtSurfaceTypeGet()` returns the type of the specified solid surface. This enables you to determine whether a surface is created by a sheet metal feature, and to distinguish among the different types of sheet metal surfaces, such as side, white, and green.

The possible values are as follows:

- PRO_SMT_SURF_NON_SMT—The surface was created by a solid feature.
- PRO_SMT_SURF_SIDE—The surface is a side surface created by a sheet metal feature.
- PRO_SMT_SURF_FACE—The surface is the face (green) surface created by a sheet metal feature.
- PRO_SMT_SURF_OFFSET—The surface is the offset (white) surface created by a sheet metal feature.

The function ProSmtedgeContourGet() returns a complete contour that contains the specified edge. This function returns PRO_TK_BAD_CONTEXT if the edge is not on the green or white side of the specified part.

The function ProSmtOppsurfGet() returns a surface that is opposite (offset to) the specified surface.

The function ProSmtOppedgeGet() returns the edge that is opposite (offset to) the specified edge. Edge data for function ProSmtOppedgeGet() uses the following definitions:

- An edge is lying on a green surface if one of its surfaces has SHEETMETAL TYPE = FACE.
- An edge is lying on a white surface if one of its surfaces has SHEETMETAL TYPE = OFFSET.
- The opposite edge to an edge must be on the surface opposite the original edge's surface and must be a geometrical offset of the original edge.
- An edge is in a peripheral contour if, and only if the following are true:
  - It is in the part geometry.
  - Exactly one of its surfaces is either FACE or OFFSET.

The function ProSmtBendsrfParentGet() returns the parent of the specified surface. For example, if the specified surface is in bent position, this function returns the surface that is the most recent, unbent equivalent of the specified surface. See notes below.

The function ProSmtBendsrfChildGet() returns the active (visible) child surface of the specified, inactive (invisible) surface. A surface is active (visible) if it is in the part geometry list. See notes below.

The function ProSmtBendedgeParentGet() returns the parent of the specified edge. For example, if the specified edge is in bent position, this function returns the edge that is the most recent, unbent equivalent of the specified edge. See notes below.

The function ProSmtBendedgeChildGet() returns the active (visible) child edge of the specified, inactive (invisible) edge. An edge is active (visible) if both its surfaces are active and the edge is contained in the contours of both surfaces. See notes below.

- Edges and surfaces in quilt geometry are also visible, but they are invalid as input to sheetmetal functions.
- Surface and edge parent and child functions use the following definitions:
  - An edge or surface has a parent if the edge or surface is a result of bending or unbending another edge or surface.
  - If an edge or surface is active and is a result of bending or unbending, any parent of this edge or surface that is in the chain of bends or unbends has this edge or surface as the active child.

The function `ProSmtMdlIsFlatStateInstance()` checks if the model is a flat state instance model.

The function `ProFaminstanceIsFlatState()` checks if the family instance of the model is a sheet metal flat instance or not.

The function `ProSmtBendsrfInfoGet()` gets all the information about the specified bend surface in a sheet metal part. You can specify as input, the face surface `PRO_SMT_SURF_FACE` or, the offset surface `PRO_SMT_SURF_OFFSET` which is created by the sheet metal feature. The cylindrical and planar surfaces, which are created by unbending the cylindrical surfaces, can be specified as input.

The following information is collected:

- *radius*—Specifies the bend radius.
- *is_inside_radius*—Specifies `PRO_B_TRUE` if the bend radius is inside. It returns `PRO_B_FALSE` if the bend radius is outside.
- *angle*—Specifies the bend angle in degrees.
- *dev_length*—Specifies the developed length of the surface.
- *dev_len_info*—Specifies a structure, that contains information about the values of various parameters, which were used to calculate the developed length. The structure `ProSmtDvlLenCalcInfo` contains the following information:
  - *method*—Specifies the method used to calculate the developed length. The method is specified using the enumerated data type `ProDvlLenMethod`.
  - *model*—Specifies the model, whose bend allowance settings are used to calculate the developed length. Usually, the model is the part that owns the specified bend surface. A model can also be a reference part, when the specified surface has been copied from a reference part. Here the developed length is calculated according to the bend allowance settings of the reference part, or the bend allowance settings of a feature in the reference part.

- *y_factor_value*—Specifies the value of K-factor or Y-factor used to calculate the developed length.

> **Note**
>
> *y_factor_value* is specified only if the method used to calculate developed length is `PRO_DVL_LEN_DRIVEN_BY_Y_FACTOR`.

- *bend_table*—Specifies the name of the bend table that controls the bend allowance calculations for the developed length.
- *formula*—Specifies the formula that was used to calculate the developed length.
- *allowance*—Specifies the value of bend allowance from the bend table.
- *dimension*—Specifies the dimension ID associated with the developed length. If the method used to calculate developed length is `PRO_DVL_LEN_DRIVEN_BY_DIMENSION`, then developed length is specified manually by the user.
- *driven_by_part_settings*—Specifies if the developed length is driven by bend allowance settings of a part or by bend allowance settings of a feature. `PRO_B_TRUE` indicates that the bend allowance settings of a part are used.

Bend Tables and Dimensions

Bend table functions support reading in or removing bent table data for a sheet metal part or feature in the part.

Sheet metal dimension functions find or set whether or not developed length dimensions are driven.

Functions Introduced:

- **ProSmtPartBendtableApply()**
- **ProSmtPartBendtableRemove()**
- **ProSmtFeatureBendtableApply()**
- **ProSmtFeatureBendtableRemove()**
- **ProSmtFeatureDevldimsGet()**
- **ProSmtDevldimIsDriven()**
- **ProSmtDevldimDrivenSet()**

The function `ProSmtPartBendtableApply()` applies the specified bent table to the sheet metal part, and then regenerates the part. The input argument *from_file* specifies whether the bend table is to be applied from memory or from the specified file.

The function `ProSmtPartBendtableRemove()` removes the specified bend table from the sheet metal part, and then regenerates the part using the Y Factor.

The function `ProSmtFeatureBendtableApply()` applies the specified bent table to the sheet metal part feature, and then regenerates the part. The input argument *from_file* specifies whether the bend table is to be applied from memory or from the specified file.

The function `ProSmtFeatureBendtableRemove()` sets a sheet metal feature to use the part bend table instead of the feature bend table, and then regenerates the part.

The function `ProSmtFeatureDevldimsGet()` returns the developed length dimensions for the specified sheet metal bend or wall feature. It also returns the surfaces whose developed length these dimensions define.

The function `ProSmtDevldimIsDriven()` specifies whether a developed length dimension is driven or not. Use the function `ProSmtDevldimDrivenSet()` to set a developed length dimension to driven.

Bend Allowance Parameters

You can set the sheet metal bend allowance properties using the bend allowance parameters. These parameters can be defined using the `ProParameter` functions. For more information on Parameters, refer to the chapter Core: Parameters on page 210.

You cannot edit these bend allowance parameters.

Update Bend Allowance from Assigned Material

Parameter Name—`SMT_UPDATE_BEND_ALLOW_INFO`

The parameter allows you to set the other bend allowance parameters to be dependent on the assigned material.

Type—Boolean

Values—Yes or No

Default Value—Yes

Bend Allowance Type

Parameter Name—`SMT_PART_BEND_ALLOW_FACTOR_TYPE`

The parameter allows you to set the bend allowance type. You can set whether the K factor or Y factor must be used.

Type—String

Values—K factor or Y factor

Default Value—Y factor

Bend Allowance Factor Value

Parameter Name—`SMT_PART_BEND_ALLOWANCE_FACTOR`

The parameter allows you to set the value of the bend allowance factor.

Type—Real number

Values—Numeric value

Default Value—0.5

Bend Allowance Table Name

Parameter Name—`SMT_PART_BEND_TABLE_NAME`

The parameter allows you to define the name of the bend allowance table.

Type—String

Values—Can be empty or list of all the names of the bend tables from the part.

Default Value—Empty

Unattached Planar Wall Feature

A planar wall is a planar section of a sheet metal part. It can either be a primary wall (the first wall in the design), or a secondary wall (which is dependent on the primary wall). Planar walls can take any flat shape.

Creo Parametric TOOLKIT supports planar walls that are created using the Fill Tool or the Planar Wall tool. Planar walls created using the Fill Tool are unattached and may be the primary wall. Wall created using the Flat Wall tool are secondary walls that are attached to existing wall edges, and may or may not have a bend applied.

Unattached Planar Wall based on the Fill Tool

A sheet metal planar wall created based on the fill tool shares most of the same elements as the standard fill feature documented in the header file `ProFlatSrf.h`. The element tree should include some of the following sheet metal specific elements to generate a sheet metal feature:

- PRO_E_IS_UNATTACHED_WALL—Has a Boolean value that specifies whether the feature is actually a flat wall.
- PRO_E_STD_DIRECTION—Specifies the material creation direction of the sheet metal flat wall, which allows you to control the thickness of the first sheet metal wall.
- PRO_E_STD_SMT_THICKNESS—Has a double value that specifies the wall thickness. If this feature is not the first wall feature in the part, the thickness value is irrelevant and can be 0.0. The feature inherits the thickness of the first wall feature. This element is not required and cannot be modified if the sheet metal thickness parameter is already assigned in the model.
- PRO_E_STD_SMT_SWAP_DRV_SIDE—Specifies whether to swap the sides of the driving and the offset surfaces (the green and white surfaces of the wall).

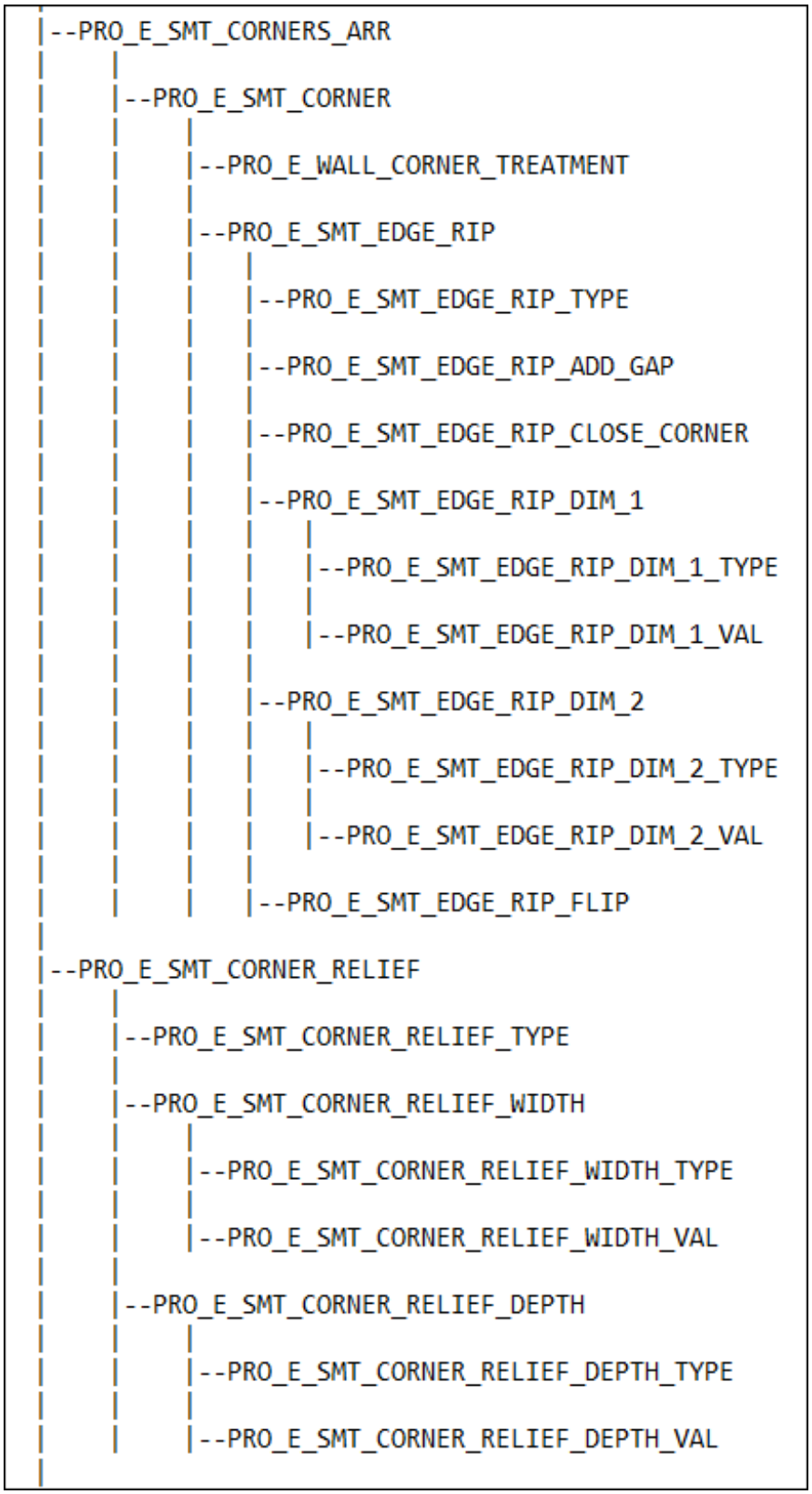
For details on standard fill features, refer to the section Fill Feature on page 859.

Feature Element Tree for the Attached Flat Wall Feature

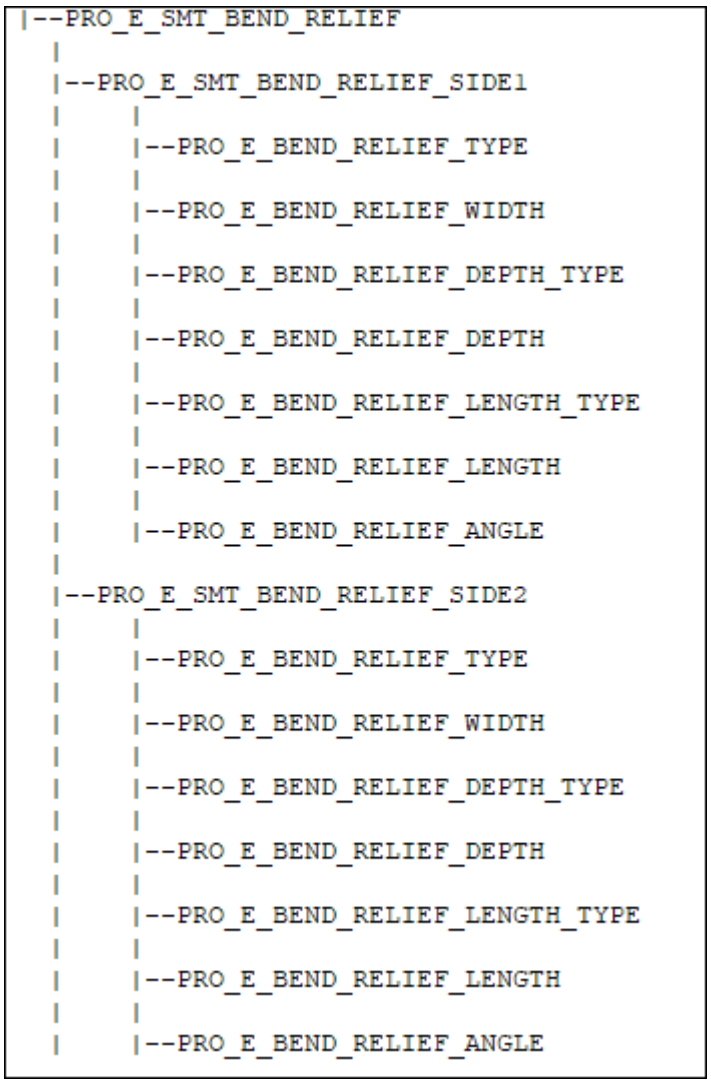
The element tree for Flat Wall feature is documented in the header file ProSmtFlatWall.h and has a simple structure. The following figure demonstrates the feature element tree structure:

Feature Element tree for Flat Wall Feature

```
PRO_E_FEATURE_TREE
      |
      |--PRO_E_FEATURE_TYPE
      |
      |--PRO_E_SMT_WALL_TYPE
      |
      |--PRO_E_STD_FEATURE_NAME
      |
      |--PRO_E_SMT_FLAT_WALL_ATT_EDGE
      |
      |--PRO_E_SMT_FLAT_WALL_ANGLE
      |
      |--PRO_E_STD_SECTION
      |
      |--PRO_E_SMT_FILLETS
      |     |
      |     |--PRO_E_SMT_FILLETS_USE_RAD
      |     |
      |     |--PRO_E_SMT_FILLETS_SIDE
      |     |
      |     |--PRO_E_SMT_FILLETS_VALUE
      |
      |--PRO_E_SMT_WALL_HEIGHT
      |     |
      |     |----PRO_E_SMT_WALL_HEIGHT_TYPE
      |     |
      |     |----PRO_E_SMT_WALL_HEIGHT_VALUE
      |
      |--PRO_E_SMT_BEND_RELIEF
      |
      |--PRO_E_SMT_WALL_THICKNESS_FLIP
      |
      |-- PRO_E_SMT_MTR_CUTS
      |      |
      |      |--PRO_E_SMT_MTR_CUTS_ADD
      |      |
      |      |--PRO_E_SMT_THREE_BEND_CRNR_RELIEF_TYPE
      |      |
      |      |--PRO_E_SMT_MITER_CUT_GROOVE_TYPE
      |      |
      |      |--PRO_E_SMT_MTR_CUTS_WIDTH_VAL
      |      |
      |      |--PRO_E_SMT_MTR_CUTS_OFFSET_VAL
      |
      |--PRO_E_SMT_DEV_LEN_CALCULATION
      |     |
      |     |--PRO_E_SMT_DEV_LEN_SOURCE
      |     |
      |     |--PRO_E_SMT_DEV_LEN_Y_FACTOR
      |     |     |
      |     |     |--PRO_E_SMT_DEV_LEN_Y_FACTOR_TYPE
      |     |     |
      |     |     |--PRO_E_SMT_DEV_LEN_Y_FACTOR_VALUE
      |     |
      |     |--PRO_E_SMT_DEV_LEN_BEND_TABLE
```


--PRO_E_SMT_CORNERS_ARR
--PRO_E_SMT_CORNER
--PRO_E_WALL_CORNER_TREATMENT
--PRO_E_SMT_EDGE_RIP
--PRO_E_SMT_EDGE_RIP_TYPE
--PRO_E_SMT_EDGE_RIP_ADD_GAP
--PRO_E_SMT_EDGE_RIP_CLOSE_CORNER
--PRO_E_SMT_EDGE_RIP_DIM_1
--PRO_E_SMT_EDGE_RIP_DIM_1_TYPE
--PRO_E_SMT_EDGE_RIP_DIM_1_VAL
--PRO_E_SMT_EDGE_RIP_DIM_2
--PRO_E_SMT_EDGE_RIP_DIM_2_TYPE
--PRO_E_SMT_EDGE_RIP_DIM_2_VAL
--PRO_E_SMT_EDGE_RIP_FLIP
--PRO_E_SMT_CORNER_RELIEF
--PRO_E_SMT_CORNER_RELIEF_TYPE
--PRO_E_SMT_CORNER_RELIEF_WIDTH
--PRO_E_SMT_CORNER_RELIEF_WIDTH_TYPE
--PRO_E_SMT_CORNER_RELIEF_WIDTH_VAL
--PRO_E_SMT_CORNER_RELIEF_DEPTH
--PRO_E_SMT_CORNER_RELIEF_DEPTH_TYPE
--PRO_E_SMT_CORNER_RELIEF_DEPTH_VAL

PRO_E_SMT_BEND_RELIEF



The feature element tree contains no non-standard element types. The following list details special information about some of the elements in this tree:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_WALL`.
- `PRO_E_SMT_WALL_TYPE`—Specifies the wall type. For Flat Walls, this should be `PRO_SMT_WALL_TYPE_FLAT`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature.
- `PRO_E_SMT_FLAT_WALL_ATT_EDGE`—Specifies the attachment edges.

- PRO_E_SMT_FLAT_WALL_ANGLE—Specifies the bend angle and consists of the following elements:
  - PRO_E_SMT_FLAT_WALL_ANGLE_TYPE specifies the angle type and is of the following types:
    - PRO_BND_ANGLE_VALUE—uses an indicated value.
    - PRO_BND_ANGLE_BY_PARAM—uses the default value of the sheet metal parameter SMT_DFLT_BEND_ANGLE.
    - PRO_BND_ANGLE_FLAT—uses no angle for the wall.
  - PRO_E_SMT_FLAT_WALL_ANGLE_VAL—specifies the angle value.
  - PRO_E_SMT_FLAT_WALL_ANGLE_FLIP—indicates whether or not to reverse the angle direction.
- PRO_E_STD_SECTION—Specifies the wall section. Wall sections can be standard or user-defined. Standard wall sections are stored in the location <creo_loadpoint>\<datecode>\Common Files\text\smt. These standard sections can be retrieved; their dimensions modified, and can be added directly into the PRO_E_STD_SECTION element tree as the PRO_E_SKETCHER element. This does not require definition of a sketch plane or viewing direction, and it does not require an incomplete feature to be created as is described in the chapter Element Trees: Sketched Features on page 1004. When standard sections are used to create the wall, the Creo Parametric user interface will show the correct type of section in the drop down menu (for example: **Rectangle**, **Trapezoid**, **L**, **T**).

If a user-defined section is to be used to create a flat wall, it must conform to the following restrictions:

- It must be a 2D section
- It must not be named the same name as one of the default section types from the Creo Parametric loadpoint.
- It must contain a horizontal centerline and 2 coordinate systems. The centerline represents the alignment with the attachment edge, and the coordinate systems represent the edge endpoints.
- The horizontal dimension must be specified.

When user-defined sections are used to create the wall, Creo Parametric automatically creates necessary sketching planes for the section during creation. Therefore the section may be assigned directly into the PRO_E_SKETCHER element without defining the sketch plane and without creating the feature as incomplete. After the feature has been created, the 3D section can be extracted from the feature element tree and the section can be modified to include references to other geometric entities in the sheet metal part.

- PRO_E_SMT_FILLETS—Specifies the bend properties of the sheet metal wall:
  - PRO_E_SMT_FILLETS_USE_RAD—true, a bend is applied, if false, no bend is used.
  - PRO_E_SMT_FILLETS_SIDE—Specifies fillet side and has the following permitted values:
    - PRO_BEND_RAD_OUTSIDE—apply the bend radius to the outside of the bend.
    - PRO_BEND_RAD_INSIDE—apply the bend radius to the inside of the bend.
    - PRO_BEND_RAD_PARAMETER—apply the bend radius at the dimension location set by the SMT_DFLT_RADIUS_SIDE parameter in Creo Parametric.
  - PRO_E_SMT_FILLETS_VALUE—Specifies bend radius.
- PRO_E_SMT_WALL_HEIGHT—Specifies the height of the attachment wall. It has the following elements:
  - PRO_E_SMT_WALL_HEIGHT_TYPE—Specifies the manner in which the newly created wall feature attaches to the attachment edge. This element takes the following values:
    - PRO_SMT_WALL_HEIGHT_AUTO—Specifies that the wall feature attaches to the attachment edge by trimming the height of the attachment wall automatically.
    - PRO_SMT_WALL_HEIGHT_VALUE—Specifies that the wall feature attaches to the attachment edge by trimming the height of the attachment wall by a specified value.
    - PRO_SMT_WALL_HEIGHT_APP_BEND—Specifies that the wall feature appends to the attachment edge without trimming the height of the attachment wall.
    - PRO_E_SMT_WALL_HEIGHT_VALUE—specifies the value of the height of the attachment wall.
    - PRO_SMT_WALL_HEIGHT_OFFSET_FROM_ORIG—Specifies that the wall feature attaches to the selected attachment edge at the specified offset distance. The distance is measured from the position of the wall, if it was attached straight to the original edge, without bend.
    - PRO_SMT_WALL_HEIGHT_OFFSET_FROM_BEND—Specifies that the wall feature appends to the selected attachment edge at the specified offset distance. The distance is measured from the position of

the wall, if it was attached to the original edge with an additional bend, as with the option PRO_SMT_WALL_HEIGHT_APP_BEND.

- PRO_E_SMT_BEND_RELIEF—Specifies bend relief at the edges of the new wall feature. The relief can be specific differently on each side of the bend:

PRO_E_SMT_BEND_RELIEF_SIDE1—Specifies the first bend relief:

- PRO_E_BEND_RELIEF_TYPE specifies relief type and has the following values:
  - PRO_BEND_RLF_NONE—specifies attachment of the wall using no relief.
  - PRO_BEND_RLF_RIP— specifies ripping of the material at each attachment point.
  - PRO_BEND_RLF_STRETCH—specifies stretching of the material for bend relief at wall attachment point.
  - PRO_BEND_RLF_RECTANGULAR—specifies adding a rectangular relief at each attachment point
  - PRO_BEND_RLF_OBROUND—specifies adding an obround relief at each attachment point.
- PRO_E_BEND_RELIEF_WIDTH—specifies the relief width (for rectangular and obround relief).
- PRO_E_BEND_RELIEF_DEPTH—specifies relief depth (for rectangular and obround relief).
- PRO_E_BEND_RELIEF_LENGTH_TYPE—specifies the relief length type and is defined by the enumerated data type ProBendRlfLengthType. The valid values follow:
  - PRO_BEND_RLF_LENGTH_NOT_USED
  - PRO_BEND_RLF_LENGTH_BLIND—Creates the bend reliefs with a length of the specified value.
  - PRO_BEND_RLF_LENGTH_TO_NEXT—Creates the bend reliefs with a length to the next surface.
  - PRO_BEND_RLF_LENGTH_THROUGH_ALL—Creates the bend reliefs through all surfaces.
  - PRO_BEND_RLF_LENGTH_TYPE_PARAM—Uses the SMT_DFLT_BEND_REL_LENGTH_TYPE parameter value.
- PRO_E_BEND_RELIEF_LENGTH—specifies the relief length value.
- PRO_E_BEND_RELIEF_ANGLE—specifies relief angle (for stretch relief).

PRO_E_SMT_BEND_RELIEF_SIDE2—Includes an identical subtree for the relief applied to the second side of the wall.

- PRO_E_SMT_WALL_THICKNESS_FLIP—Indicates whether or not to flip the thickness direction of the new wall.
- PRO_E_SMT_DEV_LEN_CALCULATION—Specifies the method used to calculate the Developed Length dimensions for bends.
- PRO_E_SMT_CORNERS_ARR—Specifies the edge transition for a particular corner intersection. See the sectionThe Element Subtree for PRO_E_SMT_CORNERS_ARR on page 1340 for more information on corner treatment.
    - PRO_E_WALL_CORNER_TREATMENT—Specifies the corner treatment that is applied to the wall. This element is defined by the enumerated data type ProWallCornerTreatment and it takes the following valid values:
        - PRO_WALL_CORNER_SEAM—Specifies if the corner is created using a seam.
        - PRO_WALL_CORNER_NO_SEAM—Specifies if the corner is created without using a seam.
        - PRO_WALL_CORNER_IGNORE—Specifies if the corner is not created.

The Element Subtree for PRO_E_SMT_MTR_CUTS

Miter Cuts

A miter cut removes material from a flat wall feature. It is controlled by two dimensions—width and offset. Offset is the distance between the end of the miter cut and the placement chain.

- PRO_E_SMT_MTR_CUTS_ADD—Specifies the miter cuts to be added.
- PRO_E_SMT_THREE_BEND_CRNR_RELIEF_TYPE—Specifies the three bend corner relief type and is defined by the enumerated data type ProThreeBendCornerType.

  ProThreeBendCornerType—Enables you to select the type of relief for the three bend corner type in a flat wall. The valid values are:
    - PRO_THREE_B_CNR_TYPE_TANGENT—Creates cut tangent in the middle bend edges to create the relief.
    - PRO_THREE_B_CNR_TYPE_CLOSED—Creates a corner tangent patch to flatten as a deformation area.
    - PRO_THREE_B_CNR_TYPE_OPEN—Creates linear cuts to the side bend vertices to create the relief.
    - PRO_THREE_B_CNR_TYPE_RIP—Creates rips to create the relief.

- PRO_THREE_B_CNR_TYPE_NO

Note

- When PRO_E_SMT_THREE_BEND_CRNR_RELIEF_TYPE is set to PRO_THREE_B_CNR_TYPE_CLOSED, the valid options for PRO_E_SMT_MITER_CUT_GROOVE_TYPE are:
  - PRO_MITER_CUT_NO_GAP
  - PRO_MITER_CUT_OBROUND
- When PRO_E_SMT_THREE_BEND_CRNR_RELIEF_TYPE is set to PRO_THREE_B_CNR_TYPE_TANGENT, PRO_THREE_B_CNR_TYPE_OPEN or PRO_THREE_B_CNR_TYPE_RIP, the valid options for PRO_E_SMT_MITER_CUT_GROOVE_TYPE are:
  - PRO_MITER_CUT_NO_GAP
  - PRO_MITER_CUT_THROUGH_ALL

- PRO_E_SMT_MITER_CUT_GROOVE_TYPE—Specifies the groove type to be cut in the miter and is defined by the enumerated data type ProMiterCutType.

  ProMiterCutType—Enables you to select the miter cut type in a flat wall. The valid values are:

  - PRO_MITER_CUT_THROUGH_ALL—Creates the miter cut groove (all the way through) to the corner relief.

    If PRO_E_SMT_MITER_CUT_GROOVE_TYPE is set to PRO_MITER_CUT_THROUGH_ALL, only PRO_E_SMT_MTR_CUTS_WIDTH_VAL is used.
  - PRO_MITER_CUT_OBROUND—Creates the miter cut groove as obround.

    If PRO_E_SMT_MITER_CUT_GROOVE_TYPE is set to PRO_MITER_CUT_OBROUND, both PRO_E_SMT_MTR_CUTS_WIDTH_VAL and PRO_E_SMT_MTR_CUTS_OFFSET_VAL are used.
  - PRO_MITER_CUT_UNDEFINED
  - PRO_MITER_CUT_NO_GAP—Creates the miter cut groove with zero width.

    If PRO_E_SMT_MITER_CUT_GROOVE_TYPE is set to PRO_MITER_CUT_NO_GAP, neither is used.

- PRO_E_SMT_MTR_CUTS_WIDTH_VAL—Specifies the width value of the miter cut.
- PRO_E_SMT_MTR_CUTS_OFFSET_VAL—Specifies the offset value of the miter cut.

The Element Subtree for Length Calculation

- PRO_E_SMT_DEV_LEN_SOURCE— Specifies the development length source. The valid values for this element are defined in the enumerated type ProDvlLenSrcType, and are as follows:
  - PRO_DVL_SRC_NOT_DEFINED— Specifies that source is not defined
  - PRO_DVL_SRC_PART_YF_AND_BTAB—uses part Y-factor and applied bend table.
  - PRO_DVL_SRC_PART_YF_ONLY—uses the part Y-factor.
  - PRO_DVL_SRC_FEAT_YF_AND_BTAB—uses the feature specific Y-factor and bend table.
  - PRO_DVL_SRC_FEAT_BTAB_ONLY—uses the feature specific bend table.
  - PRO_DVL_SRC_FEAT_YF_ONLY—uses the feature specific y-factor.
  - PRO_DVL_SRC_USE_ORIGINAL—calculates the development length using the same option which was used to create the original development length when the bend was created. For example, if original development length of the bend was calculated using a part bend table, the new development will also be calculated using the same table.
- PRO_E_SMT_DEV_LEN_Y_FACTOR—Specifies the feature Y-factor and has the following elements:
  - PRO_E_SMT_DEV_LEN_Y_FACTOR_TYPE—Specifies the types of Y-factor. The valid values for this element are defined in the enumerated type ProDvlLenFactor, and are as follows:
    - PRO_FACTOR_NOT_DEFINED
    - PRO_FACTOR_Y
    - PRO_FACTOR_K
  - PRO_E_SMT_DEV_LEN_Y_FACTOR_VALUE — Specifies the value of Y- or K- factor.
- PRO_E_SMT_DEV_LEN_BEND_TABLE — Specifies the development length bend table using the index of the bend table as loaded and stored in this model.

Note

- Bend allowance is a method used to calculate the developed length of flat sheet metal required to make a bend of a specific radius and angle. The calculation accounts for the thickness of the sheet metal, bend radii, bend angles, and other material properties such as Y- and K-factors. Developed length fluctuates with different material types and thickness, and the bend table accounts for those variations.
- Y- and K-factors are part constants defined by the location of the sheet metal material's neutral bend line. The neutral bend line position is based on a numeric reference for the type of sheet metal material used in your design. The numeric references range from 0 to 1, with the lower numbers representing softer material. Both the Y- and K-factors are integral elements in calculating the developed length (the length of flat sheet metal required to make a bend of a specific radius and angle) in your design.

Creating a Flat Wall Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Flat Wall Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Flat Wall Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Flat Wall Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Flat Wall Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Flat Wall Feature and to retrieve the element tree description of a Flat Wall Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Example 1: Creation of a Rectangular Flat Wall using a preselected edge

The sample code in `UgSmtFlatWallCreate.c` located at `creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` demonstrates how to create a rectangular flat wall using a preselected edge.

Flange Wall Feature

Flange wall features may be either swept or extruded.

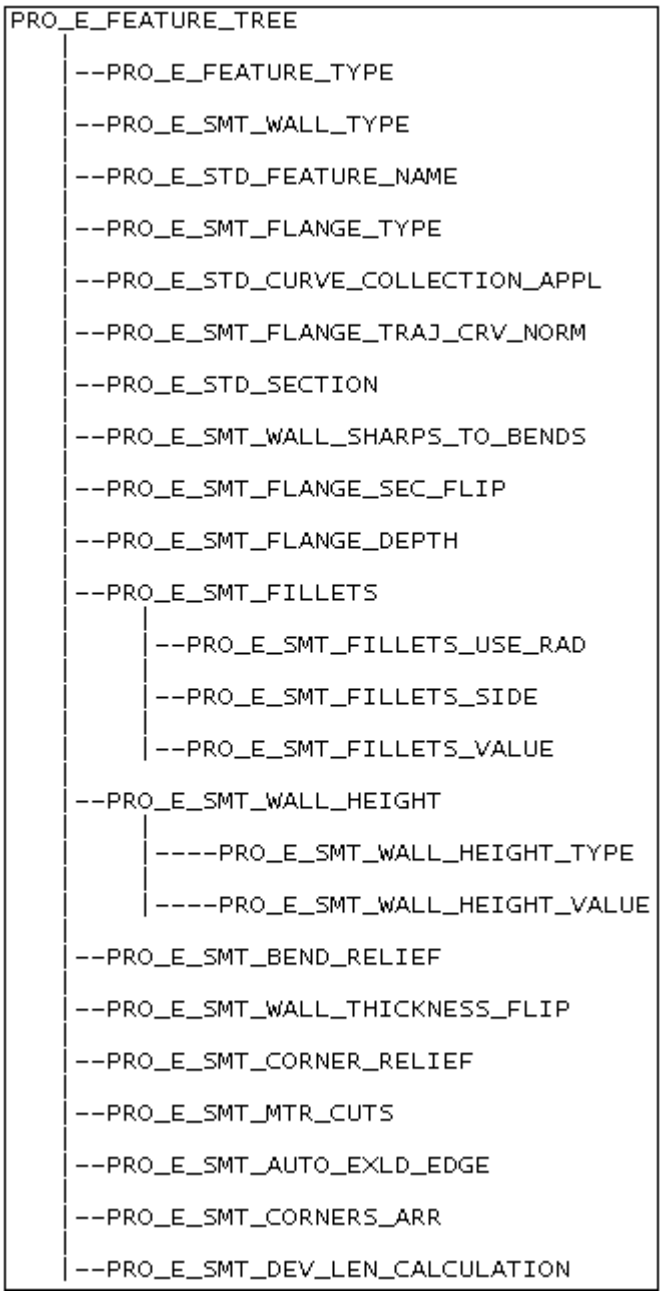
A swept flange wall follows the trajectory formed by the chain of tangent attachment edges. You can sketch a cross section along the attachment edge and the wall sweeps along that edge. The attachment edge need not be linear and the adjacent surface need not have to be planar.

An extruded flange wall extends from one linear edge into space. You can sketch the side section of the wall and project it to a certain length in both directions.

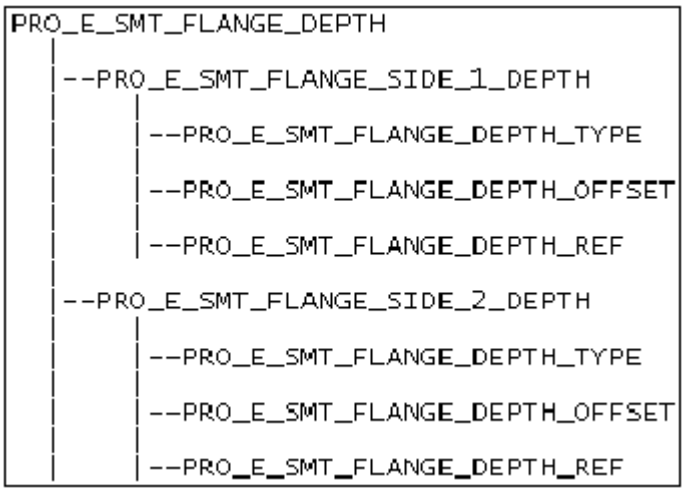
Feature Element Tree for the Flange Wall Feature

The element tree for Flange Wall feature is documented in the header file `ProSmtFlangeWall.h`, and has a simple structure. The following figure demonstrates the feature element tree structure:

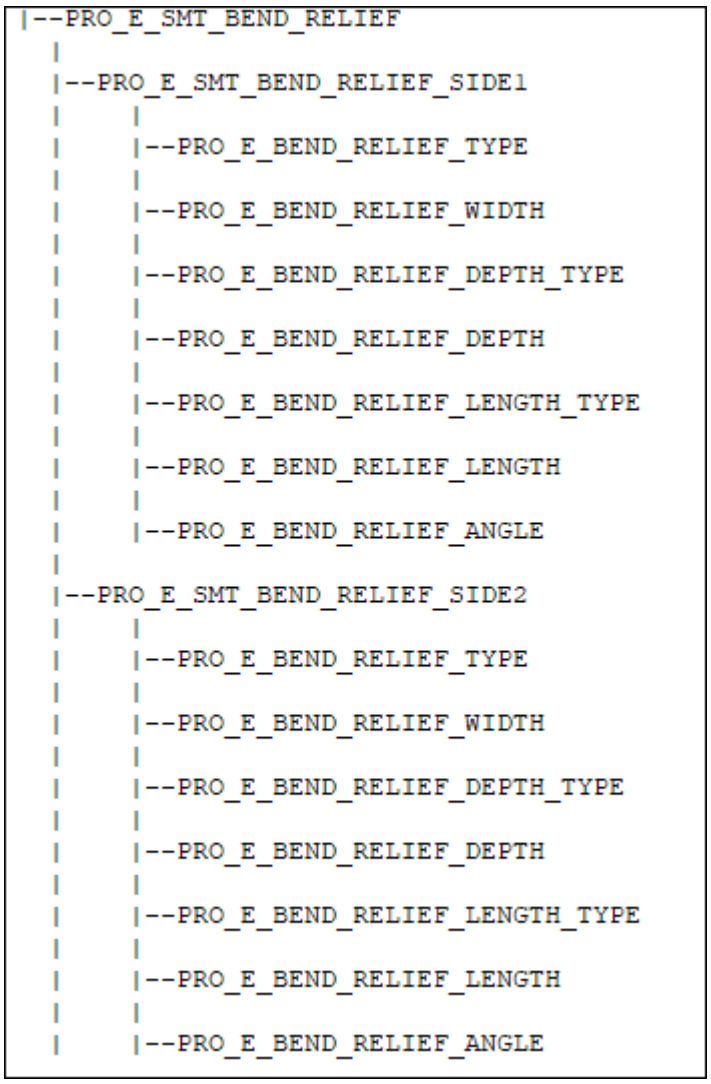
Feature Element tree for Flange Wall Feature



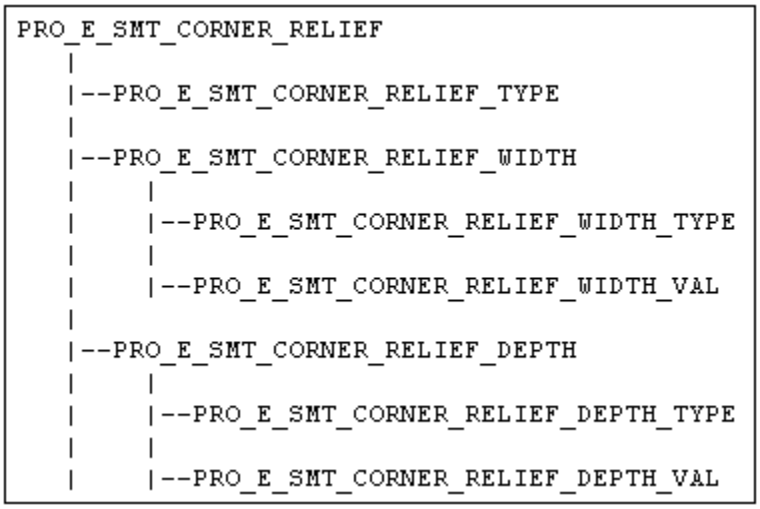
PRO_E_SMT_FLANGE_DEPTH



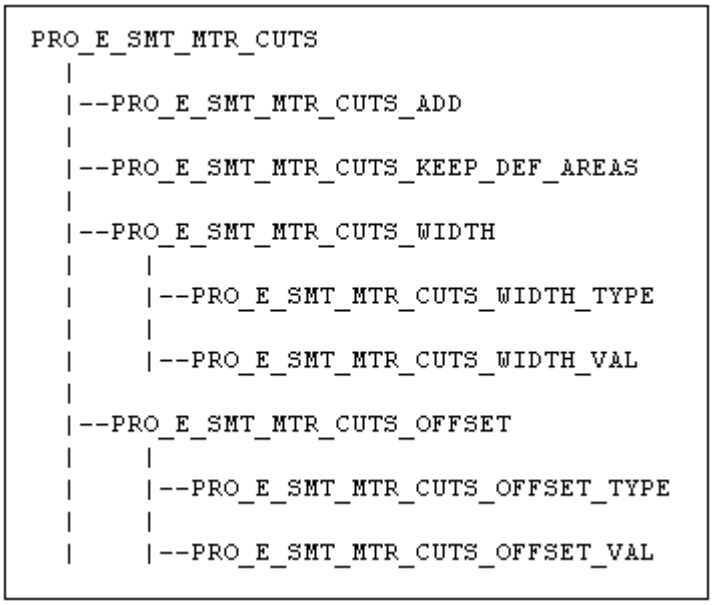
PRO_E_SMT_BEND_RELIEF



PRO_E_SMT_CORNER_RELIEF



PRO_E_SMT_MTR_CUTS



PRO_E_SMT_CORNERS_ARR

```
|--PRO_E_SMT_CORNERS_ARR
|     |
|     |--PRO_E_SMT_CORNER
|     |     |
|     |     |--PRO_E_SMT_EDGE_RIP
|     |     |    |
|     |     |    |--PRO_E_SMT_EDGE_RIP_TYPE
|     |     |    |
|     |     |    |--PRO_E_SMT_EDGE_RIP_ADD_GAP
|     |     |    |
|     |     |    |--PRO_E_SMT_EDGE_RIP_CLOSE_CORNER
|     |     |    |
|     |     |    |--PRO_E_SMT_EDGE_RIP_DIM_1
|     |     |    |    |
|     |     |    |    |--PRO_E_SMT_EDGE_RIP_DIM_1_TYPE
|     |     |    |    |
|     |     |    |    |--PRO_E_SMT_EDGE_RIP_DIM_1_VAL
|     |     |    |
|     |     |    |--PRO_E_SMT_EDGE_RIP_DIM_2
|     |     |    |    |
|     |     |    |    |--PRO_E_SMT_EDGE_RIP_DIM_2_TYPE
|     |     |    |    |
|     |     |    |    |--PRO_E_SMT_EDGE_RIP_DIM_2_VAL
|     |     |    |
|     |     |    |--PRO_E_SMT_EDGE_RIP_FLIP
```

PRO_E_SMT_DEV_LEN_CALCULATION

```
PRO_E_SMT_DEV_LEN_CALCULATION
    |
    |--PRO_E_SMT_DEV_LEN_SOURCE
    |
    |--PRO_E_SMT_DEV_LEN_Y_FACTOR
    |    |
    |    |--PRO_E_SMT_DEV_LEN_Y_FACTOR_TYPE
    |    |
    |    |--PRO_E_SMT_DEV_LEN_Y_FACTOR_VALUE
    |
    |--PRO_E_SMT_DEV_LEN_BEND_TABLE
```

Apart from the usual element for the tree root, a Flange Wall feature contains the following elements:

- PRO_E_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_WALL.
- PRO_E_SMT_WALL_TYPE—Specifies the wall type and must be
  - PRO_SMT_WALL_TYPE_FLANGE
  - PRO_SMT_WALL_TYPE_MERGE
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.
- PRO_E_SMT_FLANGE_TYPE—Specifies the type of flange wall:
  - PRO_FLANGE_WALL_TYPE_2D_SWEPT—a swept flange wall in the default orientation.
  - PRO_FLANGE_WALL_TYPE_3D_SWEPT—a swept flange wall with non-default directions, orientations, and start points.
  - PRO_FLANGE_WALL_TYPE_EXTRUDE—a flange wall extruded from a sketching plane.
- PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the attachment edge chain. If the type is swept, this can reference multiple non-tangent edges selected as one by one or using any of the other instruction types. If the flange wall type is to be "Extruded", this must contain a One by One chain with a linear edge.
- PRO_E_SMT_FLANGE_TRAJ_CRV_NORM—Specifies whether the flange wall should progress along the default edge chain direction or in the opposite direction.
- PRO_E_STD_SECTION—Specifies the wall section.

Wall sections can be standard or user-defined. Standard wall sections are stored in the location <creo_loadpoint>\<datecode>\Common Files\text\smt. These standard sections can be retrieved, their dimensions modified, and can be added directly into the PRO_E_STD_SECTION element tree as the PRO_E_SKETCHER element. This does not require definition of a sketch plane or viewing direction, and it does not require an incomplete feature to be created as is described in the chapter Creating Sketched Features. When standard sections are used to create the wall, the Creo Parametric user interface will show the correct type of section in the drop down menu (for example: **I**, **Arc**, etc.).

If a user-defined section is to be used to create a flat wall, it must conform to the following restrictions:

- It must be a 2D section.
- It must not be named the same name as one of the default section types from the Creo Parametric loadpoint.

- It must contain a horizontal centerline with a coordinate system located on it. The centerline represents the alignment with the attachment wall, and the coordinate system represents the attachment point for the section.
- The section may optionally contain bent or straight edges. It may also contain sheet metal section entities that assist in constructing the correct swept geometry.

When user-defined sections are used to create the wall, Creo Parametric automatically creates necessary sketching planes for the section during creation. Therefore the section may be assigned directly into the PRO_E_SKETCHER element without defining the sketch plane and without creating the feature as incomplete. After the feature has been created, if the wall type is 3D swept or Extruded, the 3D section can be extracted from the feature element tree and the section can be modified to include references to other geometric entities in the sheet metal part.

- PRO_E_SMT_WALL_SHARPS_TO_BENDS—If PRO_B_TRUE then Creo Parametric attempts to convert sharp edges in the section to bends.
- PRO_E_SMT_FLANGE_SEC_FLIP—Specifies whether or not to flip the direction of the section for user-defined sections.
- PRO_E_SMT_FLANGE_DEPTH—Specifies the depth of the flange, that is, the extent of the flange cover. This element governs the results for extruded flange walls only. For swept walls, the extents are governed by the rules in the element PRO_E_STD_CURV_COLLECTION_APPL, which might include trim values and boundary geometry.
- PRO_E_SMT_FILLETS—Specifies the bend properties of the sheet metal wall.
  - PRO_E_SMT_FILLETS_USE_RAD—If true, a bend is applied, if false, no bend is used.
  - PRO_E_SMT_FILLETS_SIDE—Specifies fillet side and has the following permitted values:
    - PRO_BEND_RAD_OUTSIDE—apply the bend radius to the outside of the bend.
    - PRO_BEND_RAD_INSIDE—apply the bend radius to the inside of the bend.
    - PRO_BEND_RAD_PARAMETER—apply the bend radius at the dimension location set by the SMT_DFLT_RADIUS_SIDE parameter in Creo Parametric.
  - PRO_E_SMT_FILLETS_VALUE—the bend radius.
- PRO_E_SMT_WALL_HEIGHT—Specifies the height of the attachment wall. It has the following elements:

- PRO_E_SMT_WALL_HEIGHT_TYPE—specifies the manner in which the newly created wall feature attaches to the attachment edge. This element is defined by the enumerated data type ProBendPosition and takes the following values:
  - PRO_BEND_POSITION_CONSTRAINED—Specifies that the attached wall geometry is kept within the boundary of the attachment edge.
  - PRO_BEND_POSITION_PROF_ON_EDGE—Specifies that the bend geometry is added while keeping the wall profile on the original attachment edge
  - PRO_BEND_POSITION_BEND_OUTSIDE—Specifies that the bend geometry is added with the bend line tangent to the attachment edge.
  - PRO_BEND_POSITION_OFFSET_BEND_APEX—Specifies that the offset is measured from the attachment edge to the Bend Apex.
  - PRO_BEND_POSITION_OFFSET_BEND_START—Specifies that the offset is measured from the attachment edge to the Bend Start.
- PRO_E_SMT_WALL_HEIGHT_VALUE specifies the value of the height of the attachment wall.

- PRO_E_SMT_BEND_RELIEF — Specifies bend relief. Refer to the section Feature Element Tree for the Sheetmetal Flat Wall Feature on page 1318 for more information on this element subtree.
- PRO_E_SMT_WALL_THICKNESS_FLIP — Indicates whether or not to flip the thickness direction of the new wall.
- PRO_E_SMT_CORNER_RELIEF—Indicates a compound element representing corner relief. Corner relief is added at each intersection of a pair of bends.
- PRO_E_SMT_MTR_CUTS—Indicates a compound element representing miter cuts.
- PRO_E_SMT_AUTO_EXLD_EDGE—Specifies whether to set automatic exclusion of edges. Creo Parametric uses the following set of rules and logic in order to execute the automatic wall segment excluding:
  - Long wall segment has lower excluding priority than short wall segment
  - Small wall segment that is neighbor to long wall has high excluding priority than other short wall segments
  - A wall segment whose overlapping area at the intersection of bend surfaces of neighborhood wall segments is maximum has the highest excluding priority

- PRO_E_SMT_CORNERS_ARR—Specifies the edge transitions.
- PRO_E_SMT_DEV_LEN_CALCULATION—Specifies the properties used to calculate the development length. See the section The Element Subtree for PRO_E_SMT_DEV_LEN_CALCULATION on page 1327 for more information.

The Element Subtree for PRO_E_SMT_FLANGE_DEPTH

PRO_E_SMT_FLANGE_DEPTH has the following elements:

- PRO_E_SMT_FLANGE_SIDE_1_DEPTH—Specifies first side of the flange extents and has the following elements:
  - PRO_WALL_LEN_TYPE_NONE—the flange does not extend in this direction.
  - PRO_WALL_LEN_TYPE_BLIND—the flange extends a specified length value in this direction.
  - PRO_WALL_LEN_TYPE_BLIND_SYM—the flange extends a symmetric length value in both direction. If this is used for side 1, side 2 must use PRO_WALL_LEN_TYPE_NONE.
  - PRO_WALL_LEN_TYPE_TO_REF—the flange extends to a selected geometric reference.
  - PRO_WALL_LEN_TYPE_TO_END—the flange extends to the end of the attachment reference.
  - PRO_E_SMT_FLANGE_DEPTH_OFFSET—specifies the depth offset for blind and symmetric blind extents.
  - PRO_E_SMT_FLANGE_DEPTH_REF—specifies the depth placement reference for "to ref" extents.
- PRO_E_SMT_FLANGE_SIDE_2_DEPTH—Specifies side2 the second side of the flange. This subtree is identical to the first side.

The Element Subtree for PRO_E_SMT_CORNER_RELIEF

Corner relief is required when multiple non-tangent edges are used for attachment of the flange wall. The element PRO_E_SMT_CORNER_RELIEF represents corner relief in the feature. It has the following properties:

- PRO_E_SMT_CORNER_RELIEF_TYPE specifies the types of corner reliefs:
  - PRO_CORNER_RELIEF_NO—Creo Parametric does not add relief and generates square corners.
  - PRO_CORNER_RELIEF_V_NOTCH—Creo Parametric adds a V notch shape cut at the corners.

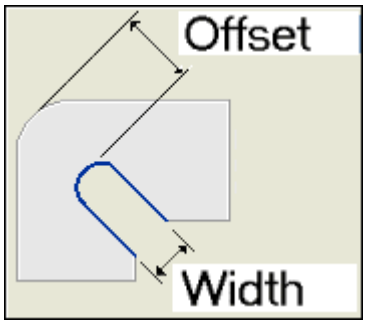
- PRO_CORNER_RELIEF_CIRCULAR—Creo Parametric adds a circular shape relief at the corners with a radius dimension.
- PRO_CORNER_RELIEF_OBROUND—Creo Parametric adds an obround relief at the corners with a specified diameter and depth.
- PRO_CORNER_RELIEF_RECTANGULAR—Creo Parametric adds a rectangular relief at the corners with a specified width and depth.

- PRO_E_SMT_CORNER_RELIEF_WIDTH a compound element with the following elements:
  - PRO_E_SMT_CORNER_RELIEF_WIDTH_TYPE—This is one of the members of ProSmdRelType. See table Relation Value Types on page 1342 for the list of value types permitted.
  - PRO_E_SMT_CORNER_RELIEF_WIDTH_VAL — This is the value for the dimension, if the width type is PRO_DIM_ENTER.
- PRO_E_SMT_CORNER_RELIEF_DEPTH a compound element with the following elements:
  - PRO_E_SMT_CORNER_RELIEF_DEPTH_TYPE—This is one of the members of ProCornerRlfDepthType. See table for the list of value types permitted.
  - PRO_E_SMT_CORNER_RELIEF_DEPTH_VAL—This is the value for the dimension, if the width type is PRO_DIM_ENTER.

The Element Subtree for PRO_E_SMT_MTR_CUTS

A miter cut removes material from any profile intersecting wall segments. The miter cut is controlled by two dimensions—width and offset as shown in the figure below. Offset is the distance between the end of the miter cut to the placement chain. Creo Parametric uses half of the specified width value to cut the material of each side of the centerline of the miter cut.

Miter Cut



It has the following elements:

- PRO_E_SMT_MTR_CUTS_ADD—Specifies whether to add miter cuts.
- PRO_E_SMT_MTR_CUTS_KEEP_DEF_AREAS—Specifies the deformation area of the miter cut. A deformation area is a section of sheet metal that helps to accurately stretch the material when you unbend the sheet metal part.
- PRO_E_SMT_MTR_CUTS_WIDTH
  - PRO_E_SMT_MTR_CUTS_WIDTH_TYPE—This is one of the members of ProMiterCutWidthType. The valid types are:
    - PRO_MITER_CUT_WIDTH_TYPE_BLIND—Specifies the type PRO_DIM_ENTER
    - PRO_MITER_CUT_WIDTH_TYPE_GAP—Specifies the type PRO_DIM_SMT_GAP
    - PRO_MITER_CUT_WIDTH_TYPE_PARAM—Specifies the type PRO_DIM_DFLT_MITER_CUT_WIDTH

    See table Relation Value Types on page 1342 for the list of value types permitted.
  - PRO_E_SMT_MTR_CUTS_WIDTH_VAL—This is the value for the dimension, if the width type is PRO_DIM_ENTER.
- PRO_E_SMT_MTR_CUTS_OFFSET
  - PRO_E_SMT_MTR_CUTS_OFFSET_TYPE—This is one of the members of ProMiterCutOffsetType. The valid types are:
    - PRO_MITER_CUT_OFFSET_TYPE_BLIND—Specifies the type PRO_DIM_ENTER
    - PRO_MITER_CUT_OFFSET_TYPE_GAP—Specifies the type PRO_DIM_SMT_GAP
    - PRO_MITER_CUT_OFFSET_TYPE_PARAM—Specifies the type PRO_DIM_DFLT_MITER_CUT_OFFSET

    See table Relation Value Types on page 1342 for the list of value types permitted.
  - PRO_E_SMT_MTR_CUTS_OFFSET_VAL—This is the value for the dimension, if the width type is PRO_DIM_ENTER.

The Element Subtree for PRO_E_SMT_CORNERS_ARR

When the flange is attached to multiple non-tangent edges, it is possible to define edge transitions for each such intersection. The members of the array element PRO_E_SMT_CORNERS_ARR each define the edge transition for a particular corner intersection. Each member has a subelement called PRO_E_SMT_EDGE_RIP which contains the following:

- PRO_E_SMT_EDGE_RIP_TYPE specifies edge treatment types and can be as follows:
  - PRO_EDGE_RIP_OPEN
  - PRO_EDGE_RIP_BLIND
  - PRO_EDGE_RIP_MITER_CUT
  - PRO_EDGE_RIP_OVERLAP
  - PRO_EDGE_RIP_CONNECT
- PRO_E_SMT_EDGE_RIP_CLOSE_CORNER specifies if the gap between the bend surfaces of a corner relief must be closed. This element is applicable only if the element PRO_E_SMT_EDGE_RIP_TYPE is set to PRO_EDGE_RIP_OPEN.
- PRO_E_SMT_EDGE_RIP_ADD_GAP specifies whether to add a gap.
- PRO_E_SMT_EDGE_RIP_DIM_1 specifies the first side's properties.
  - PRO_E_SMT_EDGE_RIP_DIM_1_TYPE — This is one of the members of ProEdgeRipDimType. The valid types are:
    - PRO_EDGE_RIP_DIM_TYPE_BLIND—Specifies the type PRO_DIM_ENTER.
    - PRO_EDGE_RIP_DIM_TYPE_GAP—Specifies the type PRO_DIM_SMT_GAP.
    - PRO_EDGE_RIP_DIM_TYPE_PARAM—Specifies the type PRO_DIM_DFLT_EDGE_TREA_WIDTH.

    See table Relation Value Types on page 1342 for the list of value types permitted.
  - PRO_E_SMT_EDGE_RIP_DIM_1_VAL — This is the value for the dimension, if the width type is PRO_DIM_ENTER.
- PRO_E_SMT_EDGE_RIP_DIM_2 specifies the second side's properties.
  - PRO_E_SMT_EDGE_RIP_DIM_2_TYPE—This is one of the members of ProEdgeRipDimType. See table Relation Value Types on page 1342 for the list of value types permitted.
  - PRO_E_SMT_EDGE_RIP_DIM_2_VAL—This is the value for the dimension, if the width type is PRO_DIM_ENTER.
- PRO_E_SMT_EDGE_RIP_FLIP specifies whether to flip the overlapping side.

Relation Value Types

ProSmdRelType	Description
PRO_DIM_THICK	The part thickness. It represents the parameter SMT_THICKNESS in a sheet metal part.
PRO_DIM_DOUBLE_THICK	2 x the part thickness. It represents the parameter SMT_THICKNESS in a sheet metal part.
PRO_DIM_ENTER	A user-defined value
PRO_DIM_DEF_CRN_REL_WIDTH	The value of the sheet metal Parameter SMT_DFLT_CRNR_REL_WIDTH, only allowed for corner relief width.
PRO_DIM_DEF_CRN_REL_DEPTH	The value of the sheet metal Parameter SMT_DFLT_CRNR_REL_DEPTH, only allowed for corner relief depth.
PRO_DIM_MINUS_THICK	-1 x the part thickness
PRO_DIM_MINUS_DOUBLE_THICK	-2 x the part thickness
PRO_DIM_DFLT_EDGE_TREA_WIDTH	The value of the sheet metal Parameter SMT_DFLT_EDGE_TREAT_WIDTH, only allowed for edge transition width
PRO_DIM_DFLT_MITER_CUT_WIDTH	The value of the sheet metal Parameter SMT_DFLT_MITER_CUT_WIDTH, only allowed for miter cut width
PRO_DIM_DFLT_MITER_CUT_OFFSET	The value of the sheet metal Parameter SMT_DFLT_MITER_CUT_OFFSET, only allowed for miter cut offset
PRO_DIM_THICK_1_1	1.1 x the part thickness
PRO_DIM_THICK_05	0.5 x the part thickness
PRO_DIM_SMT_GAP	The value of the sheet metal Parameter SMT_GAP
PRO_DIM_MINUS_SMT_GAP	The negative (minus) value of the sheet metal Parameter SMT_GAP
PRO_DIM_MINUS_THICK_05	-0.5 x the part thickness
PRO_DIM_DEF_BEND_RAD	The value of the sheet metal Parameter SMT_DFLT_BEND_RADIUS.
PRO_DIM_UP_TO_BEND	The relief depth type. Represents **Up to Bend** option for bend and corner reliefs.
PRO_DIM_TAN_TO_BEND	The relief depth type. Represents **Up to Bend** option for bend and corner reliefs.
PRO_DIM_DEF_BEND_ANGLE	The value of the sheet metal Parameter SMT_DFLT_BEND_ANGLE.
PRO_DIM_DEF_BEND_REL_WIDTH	The bend relief width. The value of the sheet metal Parameter SMT_DFLT_BEND_REL_WIDTH.
PRO_DIM_DEF_BEND_REL_DEPTH	The bend relief depth. The value of the sheet metal Parameter SMT_DFLT_BEND_REL_DEPTH.
PRO_DIM_DEF_BEND_REL_ANGLE	The bend relief angle. The value of the sheet metal Parameter SMT_DFLT_BEND_REL_ANGLE.
PRO_DIM_CRN_RLF_DEPTH_TYPE	The corner relief depth. The value of the sheet metal Parameter SMT_DFLT_CRNR_REL_DEPTH_TYPE.
PRO_DIM_BEND_RLF_DEPTH_TYPE	The bend relief depth. The value of the sheet metal Parameter SMT_DFLT_BEND_REL_DEPTH_TYPE.

Creating a Flange Wall Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Flange Wall Feature based on element tree input. For more information about `ProFeatureCreate()`, refer to the section Overview of Feature Creation on page 765 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Flange Wall Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Flange Wall Feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer to the section Feature Redefine on page 786 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Flange Wall Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to create a feature element tree that describes the contents of a Flange Wall Feature and to retrieve the element tree description of a Flange Wall Feature. For more information about `ProFeatureElemtreeExtract()`, refer to the section Feature Inquiry on page 785 in the Element Trees: Principles of Feature Creation on page 764 chapter.

Example 2: Creation of Flange Wall feature using Creo Parametric TOOLKIT

The sample code in `UgSmtFlgWallCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat` demonstrates how to create a flange wall feature using Creo Parametric TOOLKIT. The feature is created using two external properties: the wall thickness determines the bend radius, and the attachment edge(s) are obtained from the currently selected edges in the Creo Parametric model. If a single edge is selected, it is converted to a tangent chain before it used to create the feature. The section is the 2D section handle for the I-wall section provided with Creo Parametric.

Sheet metal Wall Features

In Creo Parametric TOOLKIT, you can create extruded, revolved or swept wall features. You can specify the wall thickness, generate bends, and assign a development length calculation to the wall.

An extruded wall is drawn as a cross-section extruded in the specified direction. Similarly, the revolved and swept walls are drawn.

In Creo Parametric TOOLKIT, an extruded wall shares the same element tree as the basic Extrude feature documented in the header file `ProExtrude.h`. The revolved wall shares its element tree with the basic Revolve feature documented in the header file `ProRevolve.h`. The swept protrusion wall shares its element tree with the basic sweep feature documented in documented in the header file `ProSweep.h`.

The element tree should include some of the following sheet metal-specific elements to generate a sheet metal wall feature:

- `PRO_E_STD_SMT_THICKNESS`—Has a double value that specifies the wall thickness. If this feature is not the first wall feature in the part, the thickness value is irrelevant and can be 0.0. The feature inherits the thickness of the first wall feature. This element is not required and cannot be modified if the sheet metal thickness parameter is already assigned in the model.
- `PRO_E_STD_SMT_SWAP_DRV_SIDE`—Specifies sheet metal swap sides to switch sides of driving and offset sides.
- `PRO_E_SMT_WALL_SHARPS_TO_BENDS`—Converts any sharp edges in the section to appropriate bends.
- `PRO_E_SMT_FILLETS`—Refer to the section Feature Element Tree for the Sheetmetal Flat Wall Feature on page 1318 for the description of the element.
- `PRO_E_SMT_DEV_LEN_CALCULATION`—Refer to the section Feature Element Tree for the Sheetmetal Flat Wall Feature on page 1318 for the description of the element.
- `PRO_E_SMT_MERGE_DATA`—This compound element defines the parameters required to merge the wall geometry to an existing wall.
  - `PRO_E_SMT_MERGE_AUTO`— The valid values for this element are:
    - `True`—Merges the wall geometry to an existing wall in the design.
    - `False`—Does not merge the walls.
  - `PRO_E_SMT_MERGE_KEEP_LINES`—Controls the visibility of merged edges on surface joints. The valid values for this element are:
    - `True`—Merged edges are visible on surface joints.
    - `False`—Merged edges are not visible on surface joints.

For details on the basic Extrude and Revolve features, refer to the chapter Element Trees: Extrude and Revolve on page 1013.

For details on the basic Sweep feature, refer to the chapter Element Trees: Sweep on page 1042.

Sheet metal Cut Features

A sheet metal cut removes material from the walls it encounters.

In Creo Parametric TOOLKIT, a sheet metal cut feature shares the same element tree as the basic extrude feature or the solidify feature or the thicken feature. The element tree should include some of the following sheet metal cut-specific elements to generate a sheet metal cut feature:

- `PRO_E_IS_SMT_CUT`—If true this feature is a sheet metal cut, otherwise it is a solid cut. It is applicable only in sheet metal parts.
- `PRO_E_SMT_CUT_NORMAL_DIR`—This element defines the surface to which the section projection will be normal. The projection normal specifies the side of the sheemetal wall from which the sketched curve splits the wall. The values for this element are as follows:
  - `PRO_SMT_CUT_DRVSIDE_GREEN`—Specifies the normal to the driven surface. This is a direction from the green side to the white side of the sheet metal wall. This is the default value.
  - `PRO_SMT_CUT_DRVSIDE_WHITE`—Specifies the normal to the offset surface. This is a direction from the white side to the green side of the sheet metal wall.

For details on the basic Extrude feature, see the The Element Tree for Extruded Features on page 1014.

For details on the basic Solidify feature, see the Solidify Feature on page 873.

For details on the basic Thicken feature, see the Thicken Feature on page 870.

The extrude feature has additional optional element `PRO_E_SMT_PUNCH_TOOL_DATA`. This compound element defines sheet metal cut that is used to cut and relieve sheet metal walls. It is used in sheet metal manufacturing. It is applicable to sheet metal cuts, made by the Punch UDF. It defines the parameters related to the punch feature.

The compound element `PRO_E_SMT_PUNCH_TOOL_DATA` contains the following elements:

- `PRO_E_SMT_PUNCH_TOOL_ATTR`—Specifies the symmetry flag for the Manufacturing UDF Punch Tool. The valid values for this element are:
  - `PRO_PUNCH_TOOL_ATTR_SYM_NONE`—Specifies that the tool is not symmetric about any axis.

    - PRO_PUNCH_TOOL_ATTR_SYM_X—Specifies that the tool is symmetrical about the X-axis of the coordinate system
    - PRO_PUNCH_TOOL_ATTR_SYM_Y—Specifies that the tool is symmetrical about the Y-axis of the coordinate system.
    - PRO_PUNCH_TOOL_ATTR_SYM_XY—Specifies that the tool is symmetrical about both the X and Y-axis of the coordinate system.
- PRO_E_SMT_PUNCH_TOOL_NAME—Specifies the name of the Manufacturing UDF punch tool used.

For sheet metal cuts and punches, you can specify if the punch axis point must be created. A punch axis point is a reference point that moves with a feature during both, the unbend and bend back operations. In the element PRO_E_SMT_PUNCH_AXIS_PNT, specify PRO_B_TRUE to create the punch axis point in the sheet metal feature.

Extend Wall Feature

The Extend Wall feature allows you to extend an attachment wall with a straight edge.

Feature Element Tree for the Extend Wall Feature

The element tree for the Extend Wall feature is documented in the header file ProSmtExtendWall.h. The following figure shows the feature element tree structure.

Feature Element Tree for Extend Wall Feature

```
PRO_E_FEATURE_TREE
  |
  |-- PRO_E_FEATURE_TYPE
  |
  |-- PRO_E_SMT_WALL_TYPE
  |
  |-- PRO_E_STD_FEATURE_NAME
  |
  |-- PRO_E_SMT_EXTEND_WALL_EDGE
  |
  |-- PRO_E_SMT_EXTEND_WALL_DIST
  |      |
  |      |-- PRO_E_SMT_EXTEND_DIST_TYPE
  |      |
  |      |-- PRO_E_SMT_EXTEND_DIST_VALUE
  |      |
  |      |-- PRO_E_SMT_EXTEND_DIST_REF
  |
  |-- PRO_E_SMT_EXTEND_WALL_EXTENSIONS_CMP
```

PRO_E_SMT_EXTEND_WALL_EXTENSIONS_CMP

```
PRO_E_SMT_EXTEND_WALL_EXTENSIONS_CMP
  |
  |-- PRO_E_SMT_EXTEND_SIDE1_EXTENSION_CMP
        |
        |-- PRO_E_SMT_EXTEND_EXTENSION_TYPE_OPT
        |
        |-- PRO_E_SMT_EXTEND_ADJUST_SRF

  |-- PRO_E_SMT_EXTEND_SIDE2_EXTENSION_CMP
        |
        |-- PRO_E_SMT_EXTEND_EXTENSION_TYPE_OPT
        |
        |-- PRO_E_SMT_EXTEND_ADJUST_SRF
```

The elements in this tree are described as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_WALL`.
- `PRO_E_SMT_WALL_TYPE`—Specifies the sheet metal wall type and its value is `PRO_SMT_WALL_TYPE_EXTEND`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the feature name.

- PRO_E_SMT_EXTEND_WALL_EDGE—Specifies the edge of the attachment wall you select for extension.
- PRO_E_SMT_EXTEND_WALL_DIST—Specifies the distance properties for the attachment wall to be extended. This compound element consists of the following elements:
  - PRO_E_SMT_EXTEND_DIST_TYPE—Specifies the distance type used for extension. It takes the following values
    - PRO_EXTEND_DIST_BY_VALUE—Specifies that the attachment wall is extended by a specified value and remains parallel to the selected edge.
    - PRO_EXTEND_DIST_TO_THROUGH_UNTIL—Specifies that the attachment wall is extended normally to the selected edge until it intersects the referenced plane.
    - PRO_EXTEND_DIST_TO_SELECTED—Specifies that the attachment wall is extended normally to the selected edge until it intersects the referenced plane and remains parallel to the selected edge.
  - PRO_E_SMT_EXTEND_DIST_VALUE—Specifies the distance value. This element is applicable only if the element PRO_E_SMT_EXTEND_DIST_TYPE has the value PRO_EXTEND_DIST_BY_VALUE.
  - PRO_E_SMT_EXTEND_DIST_REF—Specifies the plane or surface selected as the reference. This element is applicable only if the element PRO_E_SMT_EXTEND_DIST_TYPE has the value PRO_EXTEND_DIST_TO_THROUGH_UNTIL or PRO_EXTEND_DIST_TO_SELECTED.
- PRO_E_SMT_EXTEND_WALL_EXTENSIONS_CMP—Specifies the extension properties of the two sides of the attachment wall.

The Element Subtree for PRO_E_SMT_EXTEND_WALL_EXTENSIONS_CMP

PRO_E_SMT_EXTEND_WALL_EXTENSIONS_CMP is a compound element and consists of the following elements:

- PRO_E_SMT_EXTEND_SIDE1_EXTENSION_CMP—Specifies the extension properties of the side 1. This compound element consists of the following elements:
  - PRO_E_SMT_EXTEND_EXTENSION_TYPE_OPT—Specifies the extension type. It is given by the enumerated type ProExtendExtensionType and takes the following values:

        - PRO_EXTEND_EXT_NORMAL_TO_EDGE—Specifies that the side 1 of the attachment wall is extended normal to the selected edge.
        - PRO_EXTEND_EXT_ALONG_BOUND_EDGE—Specifies that the side 1 of the attachment wall is extended along the boundary of the selected edge.
    - PRO_E_SMT_EXTEND_ADJUST_SRF—Specifies whether the surface adjacent to side 1 of the extended edge is also extended. The values for this element are specified by the enumerated type ProExtendAdjSrf and are as follows:
        - PRO_EXTEND_ADJ_SRF_FALSE
        - PRO_EXTEND_ADJ_SRF_TRUE
- PRO_E_SMT_EXTEND_SIDE2_EXTENSION_CMP—Specifies the extension properties of the side 2. This compound element consists of the same elements as the element PRO_E_SMT_EXTEND_SIDE1_EXTENSION_CMP.

Creating a Extend Wall Feature

Function Introduced

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a Extend Wall feature based on the element tree definition. For more information about ProFeatureCreate(), refer to the Overview of Feature Creation on page 765 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Extend Wall Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function ProFeatureRedefine() to redefine a Extend Wall feature based on the changes made in the element tree. For more information about ProFeatureRedefine(), refer the Feature Redefine on page 786 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Extend Wall Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to retrieve the element tree description of the Extend Wall feature. For more information about `ProFeatureElemtreeExtract()`, refer the Feature Inquiry on page 785 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Split Area Feature

You can split a sheet metal wall using a sketched curve with the Split Area feature. When you split a sheet metal wall, no side surfaces are created.

Feature Element Tree for the Split Area Feature

The element tree for the Split Area feature is documented in the header file `ProSmtSplitArea.h`. The following figure shows the feature element tree structure.

Feature Element Tree for Split Area Feature

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_SMT_CUT_NORMAL_DIR
   |
   |--PRO_E_STD_SECTION
   |
   |--PRO_E_SMT_PROJ_DIR
   |
   |--PRO_E_STD_MATRLSIDE
```

The elements in this tree are as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_DEFORM_AREA`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature.
- `PRO_E_SMT_CUT_NORMAL_DIR`—Specifies the normal of projection. The projection normal specifies the side of the sheemetal wall from which the sketched curve splits the wall. The values for this element are as follows:
  - `PRO_SMT_CUT_DRVSIDE_GREEN`—Specifies the normal to the driven surface. This is a direction from the green side to the white side of the sheet metal wall. This is the default value.

- PRO_SMT_CUT_DRVSIDE_WHITE—Specifies the normal to the offset surface. This is a direction from the white side to the green side of the sheet metal wall.

- PRO_E_STD_SECTION—Specifies the sketch selected for the split. You can create a new section or select an internal sketch from the model.
- PRO_E_SMT_PROJ_DIR—Specifies the projection direction. It is specified by the enumerated type ProSmtProjDir. The valid values are:
  - PRO_SMT_PROJ_DIR_ONE—Specifies the projection to one side. This is the default value.
  - PRO_SMT_PROJ_DIR_TWO—Specifies the projection to the opposite side.
  - PRO_SMT_PROJ_DIR_BOTH—Specifies the projection to both the sides.
- PRO_E_STD_MATRLSIDE—Specifies the direction in which the area of the sheet metal wall is split. It is specified by the enumerated type ProSplitAreaMatSide. The valid values are:
  - PRO_SPLITAREA_MATSIDE_ONE—Specifies the split in one direction.
  - PRO_SPLITAREA_MATSIDE_TWO—Specifies the split in the opposite direction. This is the default value.

Creating a Split Area Feature

Function Introduced

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a Split Area feature based on the element tree definition. For more information about ProFeatureCreate(), refer the Overview of Feature Creation on page 765 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Split Area Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function ProFeatureRedefine() to redefine a Split Area feature based on the changes made in the element tree. For more information about ProFeatureRedefine(), refer the Feature Redefine on page 786 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Split Area Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to retrieve the element tree description of the Split Area feature. For more information about `ProFeatureElemtreeExtract()`, refer the Feature Inquiry on page 785 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Punch and Die Form Features

A Form is a sheet metal wall molded by a template (reference part). Merging the geometry of a reference part creates the Form feature.

Punch Form feature molds the sheet metal wall using only the reference part geometry whereas Die Form feature molds the sheet metal using the reference part to form the geometry (convex or concave) surrounded by a bounding plane.

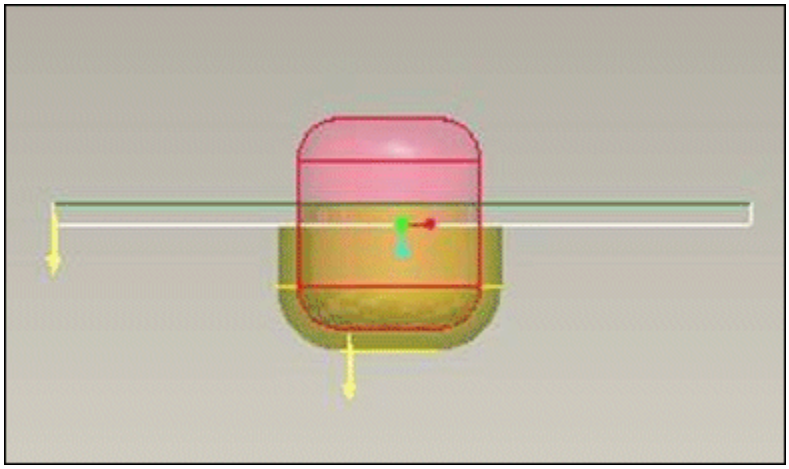
Feature Element Tree for the Punch and Die Form Features

The element tree for the Punch and Die Form features is documented in the header file `ProSmtForm.h` and can be used to create both punch and die form features. The following figure shows the feature element tree structure:

Feature Element Tree for Punch Form Feature

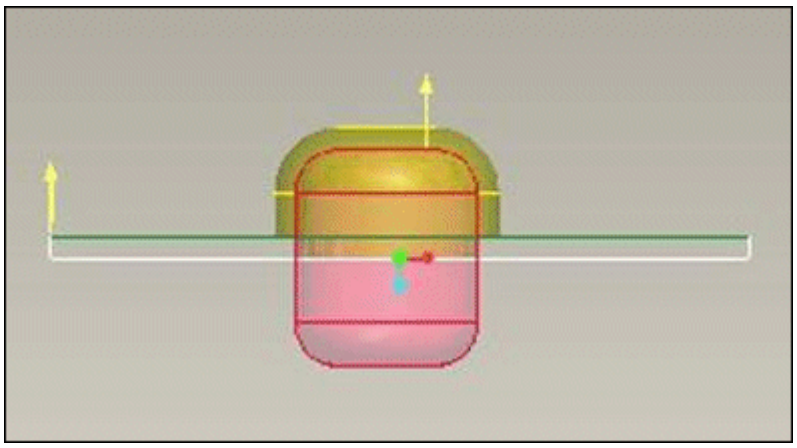
```
PRO_E_FEATURE_TREE
|
|-PRO_E_FEATURE_TYPE
|-PRO_E_STD_FEATURE_NAME
|-PRO_E_GMRG_SMT_TYPE
|-PRO_E_GMRG_FEAT_TYPE
|-PRO_E_DSF_REF_MDL
|    |--PRO_E_DSF_SEL_REF_MDL
|-PRO_E_COMP_PLACE_INTERFACE
|-PRO_E_COMPONENT_CONSTRAINTS
|-PRO_E_GMRG_VARIED_ITEMS
|-PRO_E_DSF_DEPENDENCY
|-PRO_E_FORM_PUNCH_SIDE
|-PRO_E_STD_SURF_COLLECTION_APPL
|-PRO_E_FORM_DIE_POCKET_GEOM_CMP
     |--PRO_E_STD_SURF_COLLECTION_APPL
|-PRO_E_FORM_TOOL_CSYS
|-PRO_E_FORM_TOOL_NAME
|-PRO_E_GMRG_FORM_AUTO_ROUNDS
|-PRO_E_SMT_FILLET_INTERSECT
|    |--PRO_E_SMT_FILLET_RADIUS_USEFLAG
|    |--PRO_E_SMT_FILLET_RADIUS_SIDE
|    |--PRO_E_SMT_FILLET_RADIUS_VALUE
|-PRO_E_SMT_TRIM_FORM_SIDES
```

PRO_SMT_SURF_FACE



The surface is the face (green) surface created by a sheet metal feature.

PRO_SMT_SURF_OFFSET



The surface is the offset (white) surface created by a sheet metal feature.

The elements in this tree are as follows:

- PRO_E_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_GEN_MERGE.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.
- PRO_E_GMRG_SMT_TYPE—Specifies the type of sheet metal feature. The values for this element are specified by the enumerated type ProGenMergeSmtType and the valid values are:
  - PRO_GEN_MERGE_SMT_TYPE_FORM—Specifies the Punch Form in all the versions.

    Note

    Use the element PRO_GEN_MERGE_SMT_TYPE_FORM_PUNCH instead of PRO_GEN_MERGE_SMT_TYPE_FORM from Creo Parametric3.0 onwards.

  - PRO_GEN_MERGE_SMT_TYPE_FORM_PUNCH—Specifies the Punch Form feature for parts created in Creo Parametric3.0 onward.
  - PRO_GEN_MERGE_SMT_TYPE_NOT_SMT—Specifies a merge, inheritance or any other type of General Merge feature.
  - PRO_GEN_MERGE_SMT_TYPE_FORM_DIE—Specifies the Die Form feature.
- PRO_E_GMRG_FEAT_TYPE—Specifies the type of General Merge feature.

Note

For General Merge feature of type PRO_GEN_MERGE_TYPE_MERGE, the element PRO_E_DSF_DEPENDENCY can be set to PRO_DSF_UPDATE_AUTOMATICALLY only.

For more information on the types of General Merge features, refer the General Merge (Merg on page 1211 section in the Assembly: Data Sharing Features on page 1199 chapter.

- PRO_E_DSF_REF_MDL—Specifies the punch or die model used to create the Punch or Die Form feature. It consists of the following element:
  - PRO_E_DSF_SEL_REF_MDL—Specifies the selected punch or die model.
- PRO_E_COMP_PLACE_INTERFACE—Specifies the assembly component interfaces used to define the placement of the punch or die model in the sheet metal geometry. For more information on the elements contained by the placement interfaces element, refer the Placement via Interface on page 1174 section in the Assembly: Assembling Components on page 1159 chapter.
- PRO_E_COMPONENT_CONSTRAINTS—Specifies the assembly component constraints used to define the placement of the punch or die model in the sheet metal geometry. For more information on the component constraints elements, refer the Placement Constraints on page 1172 section in the Assembly: Assembling Components on page 1159 chapter.

Note

The placement by coordinate system option in the Creo Parametric user interface for the Punch or Die Form feature is not available via Creo Parametric TOOLKIT. To place the model in the sheet metal geometry using a coordinate system, define a coordinate system feature in the sheet metal model and use it for placement.

- PRO_E_GMRG_VARIED_ITEMS—Specifies a pointer element that defines the Inheritance feature varied items and their values. This element is available only when the Punch or Die Form feature is of the type PRO_GEN_MERGE_TYPE_INHERITANCE.For more information on this element, refer the Inheritance Feature and Flexible Component Variant Items on page 1215 section in the Assembly: Data Sharing Features on page 1199 chapter.

- PRO_E_DSF_DEPENDENCY—Specifies the dependency type. The values for this element are specified by the enumerated type ProDsfDependency.

Note

From Creo Parametric 3.0 onward, the enumerated type ProDsfDependency has been deprecated. Use the enumerated type ProDSFDependency instead.

The valid values for the dependency status are:

  - PRO_DSF_UPDATE_AUTOMATICALLY—Specifies that the geometry of the Punch or Die Form feature depends upon the geometry of the model used during feature creation. The Punch or Die Form feature reflects all the changes made in the parent model.

Note

From Creo Parametric 3.0 onward, the value PRO_DSF_DEPENDENT has been deprecated. Use the enumerated value PRO_DSF_UPDATE_AUTOMATICALLY instead.

  - PRO_DSF_UPDATE_MANUALLY—Specifies that the geometry of the Punch or Die Form feature is independent of the geometry of the model used during feature creation. If you update the model, the feature does not change.

Note

From Creo Parametric 3.0 onward, the value PRO_DSF_INDEPENDENT has been deprecated. Use the enumerated value PRO_DSF_UPDATE_MANUALLY instead.

  - PRO_DSF_NO_DEPENDENCY—Specifies that there is no dependency between the geometry of the Punch or Die Form feature and the geometry of the Punch or Die model used during feature creation.

Note

All the dependency statuses specified in the enumerated type `ProDsfDependency` are defined in the header file `ProDataShareFeat.h`. For more information on the values, refer to the section Feature Element Tree on page 1211 in Assembly: Data Sharing Features.

- `PRO_E_FORM_PUNCH_SIDE`—Specifies the Punch direction. The direction specifies a side of a wall from which the model penetrates the sheet metal geometry. The values for this element are specified by the enumerated type `ProSmtSurfType`. The surface types are as follows:
  - `PRO_SMT_SURF_FACE`—The punch model moves in a direction from the green side to the white side of the sheetemetal model. Refer to the figure PRO_SMT_SURF_FACE on page 1353
  - `PRO_SMT_SURF_OFFSET`—The punch model moves in a direction from the white side to the green side of the sheet metal model. Refer to the figure PRO_SMT_SURF_OFFSET on page 1354.
- `PRO_E_STD_SURF_COLLECTION_APPL`—Specifies a collection of surfaces to be excluded from the Punch or Die model during feature creation.
- `PRO_E_FORM_DIE_POCKET_GEOM_CMP`—Compound element. Specify this element only if the enumerated type `ProGenMergeSmtType` is set to `PRO_GEN_MERGE_SMT_TYPE_FORM_DIE` type.

Note

This element cannot be used for creating Punch Form features and is specific to Die Form features only.

- `PRO_E_STD_SURF_COLLECTION_APPL`—Specifies the collection of selected surfaces to be used for Pocket Geometry during the feature creation.
- `PRO_E_FORM_TOOL_CSYS`—Specifies the reference coordinate system used during the manufacturing process. The coordinate system in the Punch model is used by default.
- `PRO_E_FORM_TOOL_NAME`—Specifies the name of the manufacturing tool used to create the Punch or Die model. The name specified by the `SMT_FORM_TOOL_NAME` parameter in the model is used by default.

- PRO_E_GMRG_FORM_AUTO_ROUNDS—Specifies the ProBoolean option to round the non-placement sharp edges (that do not lie on the placement surface). This optional element uses a constant outside radius of value equal to the thickness of the original sheet metal part.
- PRO_E_SMT_FILLET_INTERSECT—Specifies the option to round the placement sharp edges (that lie on the placement surface and intersect the sheet metal geometry). This optional element consists of the following elements:
  - PRO_E_SMT_FILLET_RADIUS_USEFLAG—Specifies whether a fillet radius is used.
  - PRO_E_SMT_FILLET_RADIUS_SIDE—Specifies the radius direction, that is, outside or inside. The values for this element, specified by the enumerated type ProSmdRadType, are as follows:
    - PRO_BEND_RAD_OUTSIDE—The radius is applied to the outside of the sheet metal geometry.
    - PRO_BEND_RAD_INSIDE—The radius is applied to the inside of the sheet metal geometry.
  - PRO_E_SMT_FILLET_RADIUS_VALUE — Specifies the radius value.
- PRO_E_SMT_TRIM_FORM_SIDES—Trim edges of sheared form. Specifies if Creo Parametric applies trimming of sheetmetal side surfaces during form feature generation. The valid values for this element follow:
  - PRO_B_TRUE
  - PRO_B_FALSE

Creating a Punch or Die Form Feature

Function Introduced

- **ProFeatureCreate()**

Use the function ProFeatureCreate() to create a Punch or Die Form feature based on the element tree definition. For more information about ProFeatureCreate(), refer the Overview of Feature Creation on page 765 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Punch or Die Form Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Punch or Die Form feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer the Feature Redefine on page 786 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Punch or Die Form Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to retrieve the element tree description of the Punch or Die Form feature. For more information about `ProFeatureElemtreeExtract()`, refer the Feature Inquiry on page 785 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

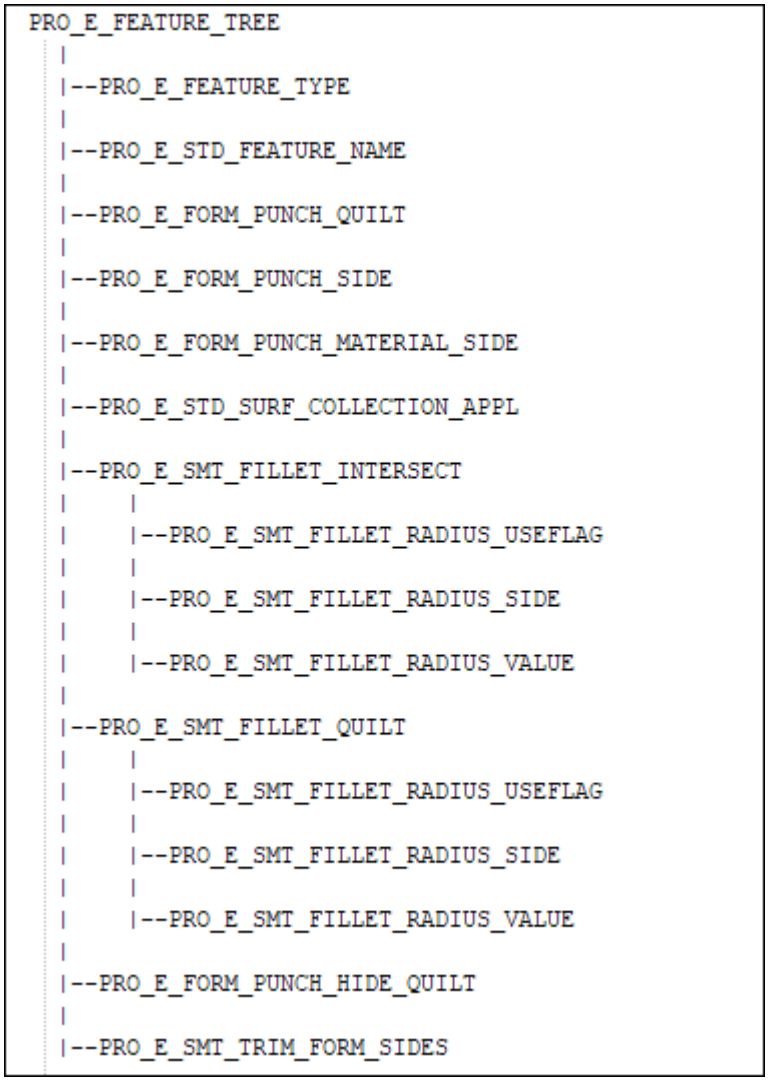
Quilt Form Feature

The Quilt Form feature molds the sheet metal wall using a referenced closed or open datum quilt.

Feature Element Tree for the Quilt Form Feature

The element tree for the Quilt Form feature is documented in the header file `ProSmtPunchQuilt.h`. The following figure shows the feature element tree structure:

Feature Element Tree for Quilt Form Feature



The elements in this tree are as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_PUNCH_QUILT`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature.
- `PRO_E_FORM_PUNCH_QUILT`—Specifies the referenced datum quilt. The geometry of the referenced quilt is merged with the sheet metal part.
- `PRO_E_FORM_PUNCH_SIDE`—Specifies the punch direction. The punch direction specifies a side of a wall from which the punch model penetrates the sheet metal geometry. The values for this element are specified by the enumerated type `ProSmtSurfType`. The surface types are as follows:

- PRO_SMT_SURF_FACE—The punch model moves in a direction from the green side to the white side of the sheetemetal model. Refer to the figure PRO_SMT_SURF_FACE on page 1353.
- PRO_SMT_SURF_OFFSET—The punch model moves in a direction from the white side to the green side of the sheet metal model. Refer to the figure PRO_SMT_SURF_OFFSET on page 1354.

- PRO_E_FORM_PUNCH_MATERIAL_SIDE — Specifies whether the resultant sheet metal geometry lies inside or outside the referenced quilt after the feature is created. The geometry is placed inside or outside the referenced quilt using the thickness value of the original sheet metal part. The values for this element, specified by the enumerated type ProSmdPunchMatSide,are as follows:
  - PRO_SMT_PUNCH_MAT_INSIDE
  - PRO_SMT_PUNCH_MAT_OUTSIDE
- PRO_E_STD_SURF_COLLECTION_APPL — Specifies a collection of surfaces to be excluded from the referenced quilt when the feature is created.
- PRO_E_SMT_FILLET_INTERSECT—Specifies the set of fillets that are added to the contours created by the intersection of the referenced quilt with the sheet metal part. This element consists of the following elements:
  - PRO_E_SMT_FILLET_RADIUS_USEFLAG—Specifies whether a fillet radius is used.
  - PRO_E_SMT_FILLET_RADIUS_SIDE—Specifies the radius direction (outside or inside). The values for this element, specified by the enumerated type ProSmdRadType, are as follows:
    - PRO_BEND_RAD_OUTSIDE—The radius is applied to the outside of the sheet metal geometry.
    - PRO_BEND_RAD_INSIDE—The radius is applied to the inside of the sheet metal geometry.
  - PRO_E_SMT_FILLET_RADIUS_VALUE—Specifies the radius value.
- PRO_E_SMT_FILLET_QUILT—Specifies the set of fillets that are added to sharps (edges between non-tangent goemetries) on the referenced quilt. This element consists of the same set of elements as PRO_E_SMT_FILLET_INTERSECT.
- PRO_E_FORM_PUNCH_HIDE_QUILT—Specifies whether the referenced quilt will be hidden in the feature. The values for this element, specified by the enumerated type ProSmdPunchHideQuilt, are as follows:
  - PRO_SMT_PUNCH_HIDE_ORIGINAL

- PRO_SMT_PUNCH_KEEP_ORIGINAL

- PRO_E_SMT_TRIM_FORM_SIDES—Trim edges of sheared form. Specifies if Creo Parametric applies trimming of sheetmetal side surfaces during form feature generation. The valid values for this element follow:
  - PRO_B_TRUE
  - PRO_B_FALSE

Creating a Quilt Form Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Quilt Form feature based on the element tree definition. For more information about `ProFeatureCreate()`, refer the Overview of Feature Creation on page 765 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Quilt Form Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Quilt Form feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer the Feature Redefine on page 786 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Quilt Form Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to retrieve the element tree description of the Quilt Form feature. For more information about`ProFeatureElemtreeExtract()`, refer the Feature Inquiry on page 785 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

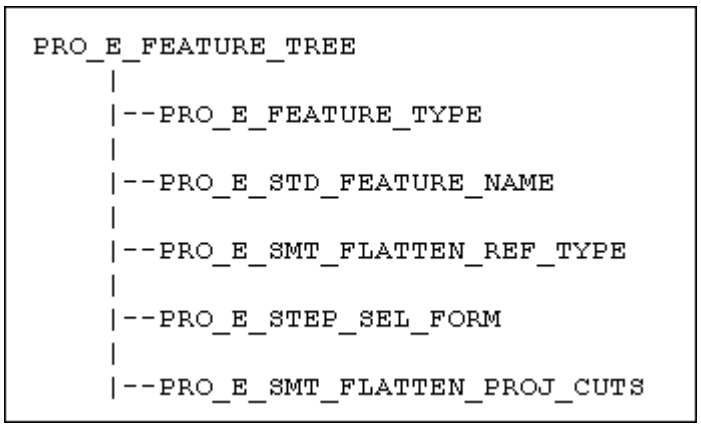
Flatten Form Feature

The Flatten Form feature allows you to flatten existing Form features in your model.

Feature Element Tree for Flatten Form Feature

The element tree for a Flatten Form feature is documented in the header file `ProSmtFlattenForm.h` and has a simple structure. The following figure demonstrates the structure of the feature element tree.

Flatten Form Element Tree



The elements in this tree are described as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_FLATTEN`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature.
- `PRO_E_SMT_FLATTEN_REF_TYPE`—Specifies the selection type for the Form features to be flattened. It is specified by the enumerated type `ProFlattenRefType`. The valid selection types are:
  - `PRO_FLATTEN_FORM_REFSEL`—Specifies the type where you select an array of Form features from the model. This is the default value.
  - `PRO_FLATTEN_FORM_ALLSEL`—Specifies the type where Creo Parametric finds and selects all the Form features from the model.
- `PRO_E_STEP_SEL_FORM`—Specifies the array of Form features you select.
- `PRO_E_SMT_FLATTEN_PROJ_CUTS`—Specifies if cuts must be projected to the flattened form.

Creating a Flatten Form Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a Flatten Form feature based on the element tree definition. For more information about `ProFeatureCreate()`, refer the Overview of Feature Creation on page 765 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Flatten Form Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Flatten Form feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer the Feature Redefine on page 786 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Flatten Form Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to retrieve the element tree description of the Flatten Form feature. For more information about `ProFeatureElemtreeExtract()`, refer the Feature Inquiry on page 785 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

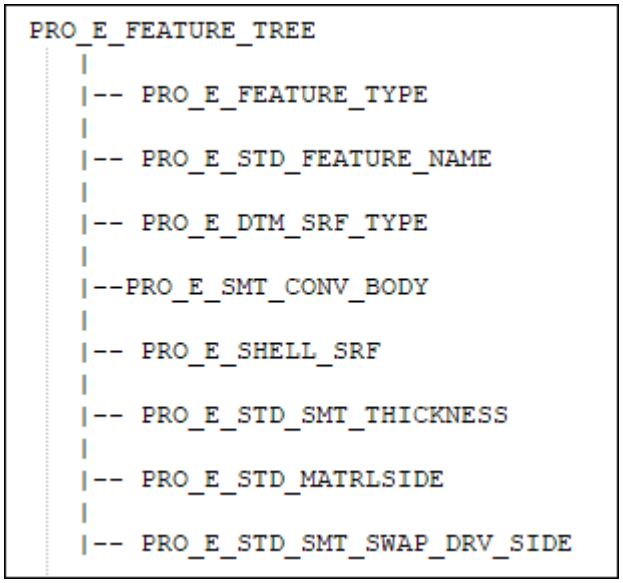
Convert Features

Shell Feature

You can use the convert features to convert a solid part into a sheet metal part. For a block-like part, use the Shell feature to remove one or more walls that hollows the inside of the model, leaving a shell of the specified wall thickness.

The element tree for the Shell feature is documented in the header file `ProSmtShell.h`, and is shown in the following figure:

Element Tree for Shell Feature



The elements in this tree are described as follows:

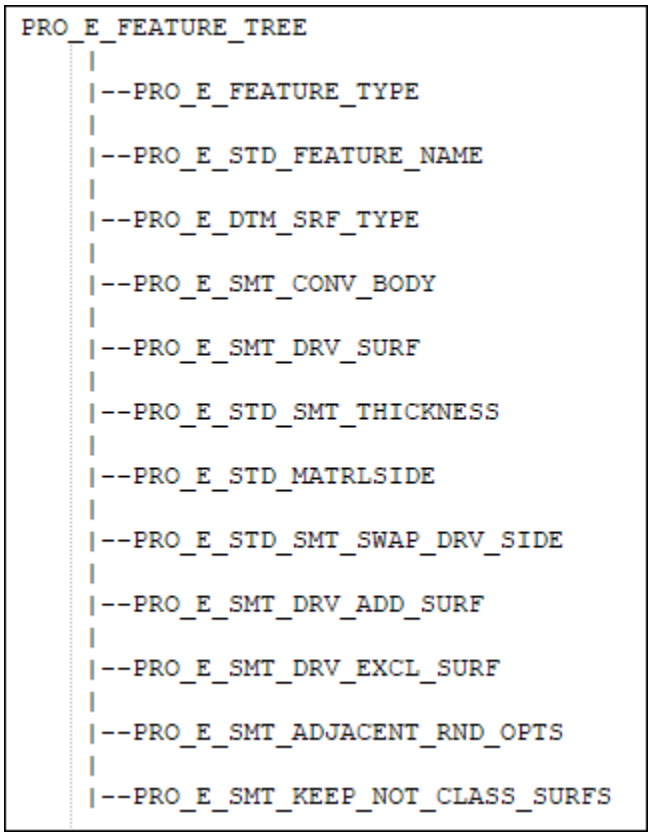
Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature. The value of this feature must be PRO_FEAT_DATUM_SURF.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Shell_1.
PRO_E_DTM_SRF_TYPE	PRO_VALUE_TYPE_INT	Specifies the Datum Surface Type using the enumerated type ProSmtDtmSrfType. The value of this feature must be PRO_DTM_SRF_AS_WALL_SHELL.
PRO_E_SMT_CONV_BODY	PRO_VALUE_TYPE_SELECTION	Specifies the body to be selected to convert to a sheetmetal part.
PRO_E_SHELL_SRF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the collection of surfaces to be removed to shell the solid.
PRO_E_STD_SMT_THICKNESS	PRO_ELEM_TYPE_DOUBLE	Specifies the thickness of the wall. It must be positive a number.
PRO_E_STD_MATRLSIDE	PRO_VALUE_TYPE_INT	Specifies the direction of material thickness.
PRO_E_STD_SMT_SWAP_DRV_SIDE	PRO_VALUE_TYPE_INT	Specifies sheet metal swap sides to switch sides of shelled and driving surfaces.

Driving Surface Feature

The Driving Surface feature converts a solid geometry to sheet metal part. For thin protrusions with constant thickness, use the Driving Surface feature to select a surface as the driving surface and to set the wall thickness.

The element tree for the Driving Surface feature is documented in the header file `ProSmtDrvSurf.h` and is shown in the following figure:

Element Tree for Driving Surface Feature



The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature. The value of this feature must be PRO_FEAT_DATUM_SURF.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Driving_Surface_1.
PRO_E_DTM_SRF_TYPE	PRO_VALUE_TYPE_INT	Specifies the Datum Surface Type using the enumerated type ProSmtDtmSrfType. The value of this feature must be PRO_DTM_SRF_AS_WALL. For an empty body, the value of this

Element ID	Data Type	Description
		feature must be PRO_DTM_SRF_EMPTY_BODY_CONV.
PRO_E_SMT_CONV_BODY	PRO_VALUE_TYPE_SELECTION	Specifies the body to be selected to be converted to a sheetmetal part.
PRO_E_SMT_DRV_SURF	PRO_VALUE_TYPE_SELECTION	Specifies the collection of a solid surface to be used as the driving surface.
PRO_E_STD_SMT_THICKNESS	PRO_ELEM_TYPE_DOUBLE	Specifies the thickness of the wall.
PRO_E_STD_MATRLSIDE	PRO_VALUE_TYPE_INT	Specifies the direction of material thickness.
PRO_E_STD_SMT_SWAP_DRV_SIDE	PRO_VALUE_TYPE_INT	Specifies sheet metal swap sides to switch sides of driving and selected surfaces.
PRO_E_SMT_DRV_ADD_SURF	Compound	Specifies the additional surface to be used as a driving surface for the sheetmetal body part.
PRO_E_SMT_DRV_EXCL_SURF	Compound	Specifies the surfaces to be excluded so that they are not treated as face surfaces in the sheet metal body.

Element ID	Data Type	Description
PRO_E_SMT_ADJACENT_RND_OPTS	PRO_VALUE_TYPE_INT	Specifies the adjacent round options and is defined by the enumerated date type ProSmtAdjRndOpts. The valid values follow: • PRO_SMT_ADJ_RND_RECREATE—Removes rounds and chamfers and recreates them after the part is converted to a sheet metal part. • PRO_SMT_ADJ_RND_REMOVE—Removes the rounded geometry. The resulting geometry is similar to the geometry before the round operation. • PRO_SMT_ADJ_RND_IGNORE—Ignores the rounded geometry. The resulting geometry is without the rounded geometry.
PRO_E_SMT_KEEP_NOT_CLASS_SURFS	PRO_VALUE_TYPE_INT	Specifies the keep not classified surfaces and is defined by the enumerated data type ProSmtKeepNotClassSurfsType. The valid values follow: • PRO_SMT_IGNORE_NOT_CLASS_SURFS —Ignores surfaces that are not classified as the driving surface, additional surfaces, and excluded surfaces as separate quilts. • PRO_SMT_KEEP_NOT_CLASS_SURFS—Keeps surfaces that are not classified as the driving surface, additional surfaces, and excluded surfaces as separate quilts.

Rip Features

Rip features allow you to tear a continuous piece of sheet metal material so that when you unbend a design, it tears along the ripped section. There are four types of rips available:

- `Sketched Rip`—Tears the sheet metal part along a sketched path. You can exclude surfaces to protect them from rip.
- `Surface Rip`—Cuts out a surface patch from the sheet metal part and in the process removes volume from the part.
- `Edge Rip`—Tears the sheet metal part along an edge. You can define edge treatment for the ripped edges.
- `Rip Connect`—Tears the sheet metal part between two datum points or vertices or a combination of both.

The four rip types are specified by the enumerated type `ProSmtRipType` and take the following values:

- `PRO_SMT_RIP_REGULAR`—Specifies a Sketched Rip.
- `PRO_SMT_RIP_SURFACE`—Specifies a Surface Rip.
- `PRO_SMT_RIP_EDGE`—Specifies an Edge Rip.
- `PRO_SMT_RIP_CONNECT`—Specifies a Connect Rip.

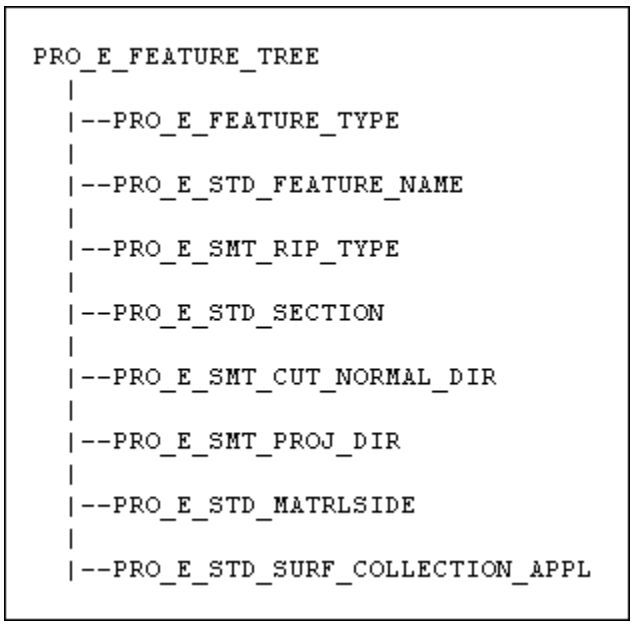
Sketched Rip Feature

The Sketched Rip feature allows you to tear the sheet metal part along a sketched path.

Feature Element Tree for Sketched Rip Feature

The element tree for a Sketched Rip feature is documented in the header file `ProSmtRegularRip.h` and has a simple structure. The following figure demonstrates the structure of the feature element tree.

Sketched Rip Element Tree



The elements in this tree are described as follows:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type and should be `PRO_FEAT_RIP`.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the feature.
- `PRO_E_SMT_RIP_TYPE`—Specifies the rip type and should have the value `PRO_SMT_RIP_REGULAR`.
- `PRO_E_STD_SECTION`—Specifies the sketch selected for the rip. You can create a new section or select an internal sketch from the model.
- `PRO_E_SMT_CUT_NORMAL_DIR`—Specifies the normal of projection. The projection normal specifies the side of the sheemetal part from where the sketched curve rips the part. The values for this element are as follows:
  - `PRO_SMT_CUT_DRVSIDE_GREEN`—Specifies the normal to the driven surface. This is a direction from the green side to the white side of the sheet metal part. This is the default value.
  - `PRO_SMT_CUT_DRVSIDE_WHITE`—Specifies the normal to the offset surface. This is a direction from the white side to the green side of the sheet metal part.
- `PRO_E_SMT_PROJ_DIR`—Specifies the projection direction. It is specified by the enumerated type `ProSmtProjDir`. The valid values are:
  - `PRO_SMT_PROJ_DIR_ONE`—Specifies the projection to one side. This is the default value.

- PRO_SMT_PROJ_DIR_TWO—Specifies the projection to the opposite side.
- PRO_SMT_PROJ_DIR_BOTH—Specifies the projection to both the sides.

- PRO_E_STD_MATRLSIDE—Specifies the direction in which the area of the sheet metal part is ripped. It is specified by the enumerated type ProSketchRipMatSide. The valid values are:
  - PRO_SKETCHRIP_MATSIDE_ONE—Specifies the rip in one direction.
  - PRO_SKETCHRIP_MATSIDE_TWO—Specifies the rip in the opposite direction. This is the default value.
- PRO_E_STD_SURF_COLLECTION_APPL—Specifies a collection of surfaces that are excluded from the rip operation. This element is optional.

Surface Rip Feature

The Surface Rip feature allows you to cut out a surface patch from the sheet metal part and in the process removes volume from the part.

Feature Element Tree for Surface Rip Feature

The element tree for a Surface Rip feature is documented in the header file ProSmtSurfaceRip.h and has a simple structure. The following figure demonstrates the structure of the feature element tree.

Surface Rip Element Tree

```
PRO_E_FEATURE_TREE
  |
  |--PRO_E_FEATURE_TYPE
  |
  |--PRO_E_STD_FEATURE_NAME
  |
  |--PRO_E_SMT_RIP_TYPE
  |
  |--PRO_E_STD_SURF_COLLECTION_APPL
```

The elements in this tree are described as follows:

- PRO_E_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_RIP.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.

- PRO_E_SMT_RIP_TYPE—Specifies the rip type and should have the value PRO_SMT_RIP_SURFACE.
- PRO_E_STD_SURF_COLLECTION_APPL—Specifies the collection of surfaces you select to be ripped.

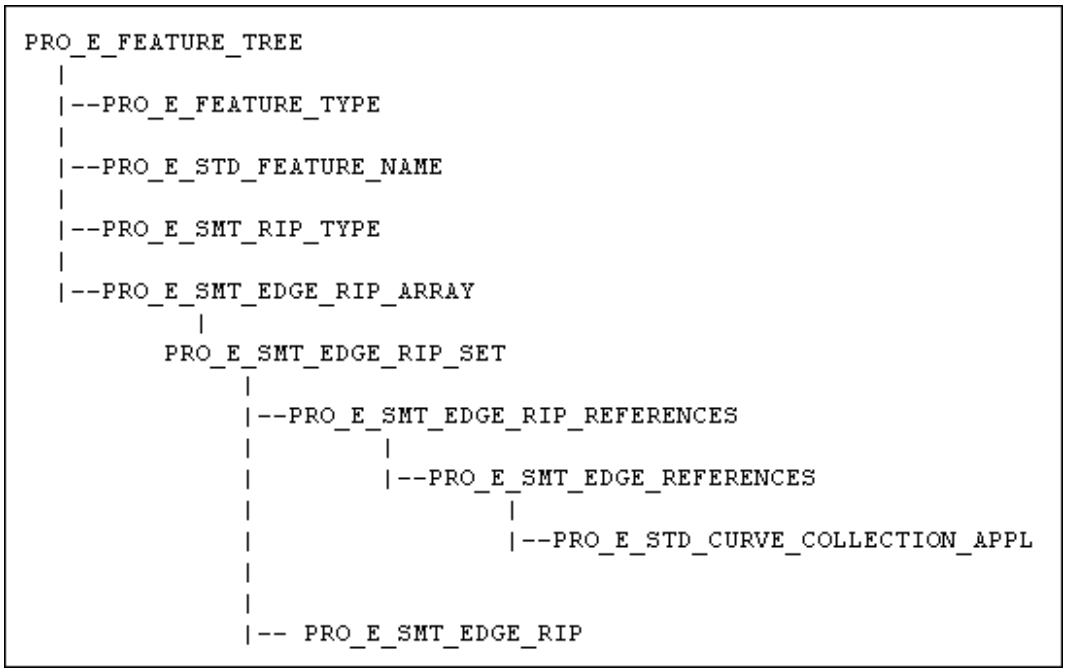
Edge Rip Feature

The Edge Rip feature allows you to tear the sheet metal part along an edge.

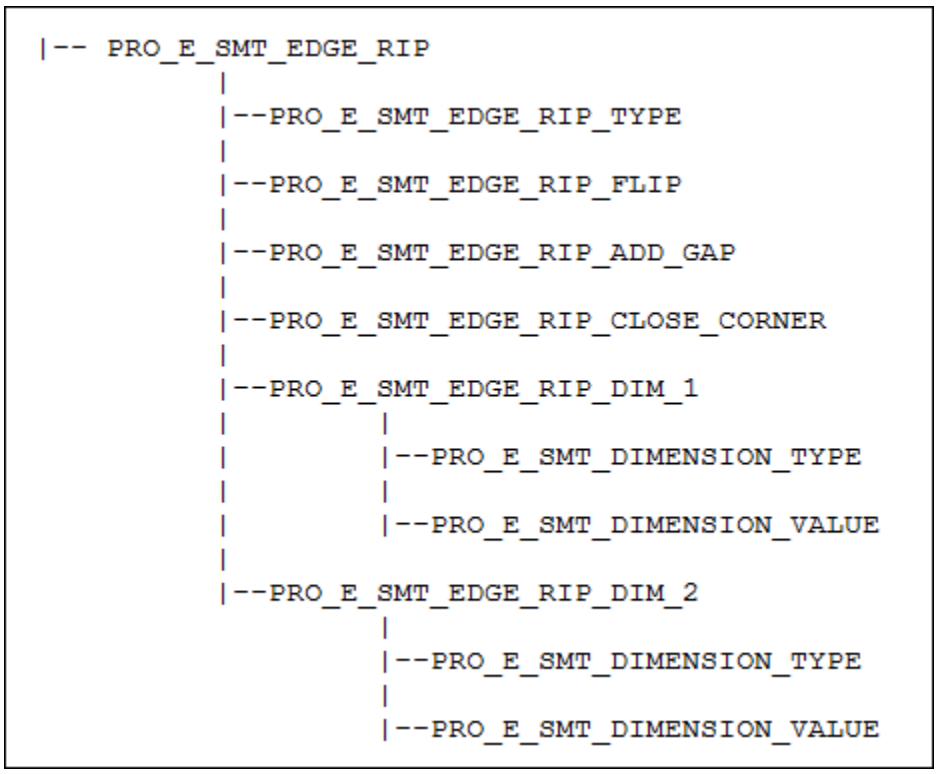
Feature Element Tree for Edge Rip Feature

The element tree for a Edge Rip feature is documented in the header file ProSmtEdgeRip.h and has a simple structure. The following figure demonstrates the structure of the feature element tree.

Edge Rip Element Tree



PRO_E_SMT_EDGE_RIP



The elements in this tree are described as follows:

- PRO_E_FEATURE_TYPE—Specifies the feature type and should be PRO_FEAT_RIP.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the feature.
- PRO_E_SMT_RIP_TYPE—Specifies the rip type and should have the value PRO_SMT_RIP_EDGE.
- PRO_E_SMT_EDGE_RIP_ARRAY—Specifies an array of the element PRO_E_SMT_EDGE_RIP_SET. PRO_E_SMT_EDGE_RIP_SET is a compound element that consists of the following elements:
  - PRO_E_SMT_EDGE_RIP_REFERENCES—Specifies an array of the element PRO_E_SMT_EDGE_REFERENCES. PRO_E_SMT_EDGE_REFERENCES is a compound element that consists of the following element:
    - PRO_E_STD_CURVE_COLLECTION_APPL—Specifies the chain of edges selected for the rip.
  - PRO_E_SMT_EDGE_RIP—Specifies the types and options for the treatment of the ripped edges.

The Element Subtree for PRO_E_SMT_EDGE_RIP

PRO_E_SMT_EDGE_RIP is a compound element that consists of the following elements:

- PRO_E_SMT_EDGE_RIP_TYPE—Specifies the edge treatment types. It is specified by the enumerated type ProEdgeRipType. The valid types are:
  - PRO_EDGE_RIP_OPEN—Rips the sheet metal walls at their point of intersection.
  - PRO_EDGE_RIP_BLIND—Rips the sheet metal part with a gap specified by two dimensions.
  - PRO_EDGE_RIP_MITER_CUT—Rips the sheet metal part with a gap specified by a single dimension.
  - PRO_EDGE_RIP_OVERLAP—Rips the sheet metal part such that one side overlaps the other.
  - PRO_EDGE_RIP_PARAM—Rips the sheet metal part by the value specified by the defined SMT_GAP parameter.
- PRO_E_SMT_EDGE_RIP_FLIP—Specifies whether to flip the overlapping side. This element is available only if the PRO_E_SMT_EDGE_RIP_TYPE has the value PRO_EDGE_RIP_OVERLAP.
- PRO_E_SMT_EDGE_RIP_ADD_GAP—Specifies whether to a add a gap clearance. This element is applicable only if the element PRO_E_SMT_EDGE_RIP_TYPE has the value PRO_EDGE_RIP_OVERLAP or PRO_EDGE_RIP_PARAM.
- PRO_E_SMT_EDGE_RIP_CLOSE_CORNER—Specifies if the gap between the bend surfaces of an edge rip must be closed. This element is applicable only if the element PRO_E_SMT_EDGE_RIP_TYPE is set to PRO_EDGE_RIP_OPEN.
- PRO_E_SMT_EDGE_RIP_DIM_1—Specifies the properties of side 1. This element consists of the following elements:
  - PRO_E_SMT_DIMENSION_TYPE—Specifies the dimension type. It is specified by the enumerated type ProEdgeRipDimType. The valid types are:
    - PRO_EDGE_RIP_DIM_TYPE_BLIND—Specifies the type PRO_DIM_ENTER.
    - PRO_EDGE_RIP_DIM_TYPE_GAP—Specifies the type PRO_DIM_SMT_GAP.
    - PRO_EDGE_RIP_DIM_TYPE_PARAM—Specifies the type PRO_DIM_DFLT_EDGE_TREA_WIDTH.

See table Relation Value Types on page 1342 for the descriptions of the above list of value types.

- PRO_E_SMT_DIMENSION_VALUE—Specifies the dimension value for the type selected.

- PRO_E_SMT_EDGE_RIP_DIM_2—Specifies the properties of side 2. This compound element consists of the same elements as the element PRO_E_SMT_EDGE_RIP_DIM_1.

Rip Connect Feature

The Rip Connect feature allows you to tear the sheet metal part between two datum points or vertices or a combination of both. A rip connect endpoint must be either a datum point, or a vertex and must lie at the end of an edge rip or on the part border.

The element tree for the Rip Connect feature is documented in the header file ProSmtConnectRip.h and is as shown in the following figure:

Element Tree for Rip Connect Feature

```
PRO_E_FEATURE_TREE
  |
  |--PRO_E_FEATURE_TYPE
  |
  |--PRO_E_STD_FEATURE_NAME
  |
  |--PRO_E_SMT_RIP_TYPE
  |
  |--PRO_E_SMT_RIP_CONNECT_COMP
        |
        |--PRO_E_SMT_RIP_CONNECT_SETS
              |
              |--PRO_E_SMT_RIP_CONNECT_SET
                    |
                    |--PRO_E_SMT_RIP_CONN_REFS
                    |      |
                    |      |--PRO_E_SMT_RIP_CONN_REF
                    |            |
                    |            |--PRO_E_SMT_RIP_CONN_END
                    |
                    |--PRO_E_SMT_RIP_CONN_GAP_COMP
                          |
                          |--PRO_E_SMT_RIP_CONN_ADD_GAP
                          |
                          |--PRO_E_SMT_RIP_CONN_GAP_VAL
```

The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature. The value of this feature must be PRO_FEAT_RIP.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Rip_Connect_1.
PRO_E_SMT_RIP_TYPE	PRO_VALUE_TYPE_INT	Specifies the rip type. The valid value for this element is PRO_SMT_RIP_CONNECT.
PRO_E_SMT_RIP_CONNECT_COMP	Compound	Specifies a compound element.
PRO_E_SMT_RIP_CONNECT_SETS	Array	Specifies an array of rip connect sets.
PRO_E_SMT_RIP_CONNECT_SET	Compound	Specifies a compound element of reference points and gap parameters.
PRO_E_SMT_RIP_CONN_REFS	Array	Specifies an array element containing the starting and end point of the rip connect feature. You can specify up to two elements in this array.
PRO_E_SMT_RIP_CONN_REF	Compound	Specifies a compound element of vertex or datum points.
PRO_E_SMT_RIP_CONN_END	PRO_VALUE_TYPE_SELECTION	Specifies a vertex or datum point to define the start or end of the rip. This vertex or datum point must be placed on the edge or the border of a sheet metal part.
PRO_E_SMT_RIP_CONN_GAP_COMP	Compound	Specifies a compound element of gap parameters.
PRO_E_SMT_RIP_CONN_ADD_GAP	PRO_VALUE_TYPE_BOOLEAN	Specifies if a gap clearance should be added to the selected set of the rip connect. The valid values are TRUE or FALSE.
PRO_E_SMT_RIP_CONN_GAP_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the gap value.

Creating a Rip Feature

Function Introduced

- **ProFeatureCreate()**

Use the function `ProFeatureCreate()` to create a specific Rip feature based on the element tree definition. For more information about `ProFeatureCreate()`, refer the Overview of Feature Creation on page 765 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Redefining a Rip Feature

Function Introduced

- **ProFeatureRedefine()**

Use the function `ProFeatureRedefine()` to redefine a Rip feature based on the changes made in the element tree. For more information about `ProFeatureRedefine()`, refer the Feature Redefine on page 786 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Accessing a Rip Feature

Function Introduced

- **ProFeatureElemtreeExtract()**

Use the function `ProFeatureElemtreeExtract()` to retrieve the element tree description of the Rip feature. For more information about `ProFeatureElemtreeExtract()`, refer the Feature Inquiry on page 785 section in the Element Trees: Principles of Feature Creation on page 764 chapter.

Corner Relief Feature

Corner relief can be added at each intersection of a pair of bends. When you add relief, sheet metal sections are removed from the model.

The element tree for the Corner Relief feature is documented in the header file `ProSmtCornerRelief.h` and is shown in the following figure:

Element Tree for Corner Relief Feature

```
PRO_E_FEATURE_TREE
   |
   |-- PRO_E_STD_FEATURE_NAME
   |
   |-- PRO_E_CORNER_RELIEFS (compound)
        |
        |-- PRO_E_CORNER_RELIEFS_CR_STATE (option)
        |
        |-- PRO_E_CORNER_RELIEFS_ARR (array)
         |
         |-- PRO_E_CORNER_RELIEF_SET (compound)
              |
              |-- PRO_E_CORNER_RELIEF_REF_ARR (array)
              |    |
              |    |-- PRO_E_CORNER_RELIEF_REF_SET (compound)
              |         |
              |         |-- PRO_E_CORNER_RELIEF_REF_TYPE (option)
              |         |
              |         |-- PRO_E_CORNER_RELIEF_REF_FLAT (geom)
              |         |
              |         |-- PRO_E_CORNER_RELIEF_REF_BND_1 (geom)
              |         |
              |         |-- PRO_E_CORNER_RELIEF_REF_BND_2 (geom)
              |
              |-- PRO_E_CORNER_RELIEF_DEFINE (compound)
```

The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Corner_Relief_1.
PRO_E_CORNER_RELIEFS	Compound	This compound element defines the corner relief sets.
PRO_E_CORNER_RELIEFS_CR_STATE	PRO_VALUE_TYPE_INT	Specifies the corner relief state using the enumerated data type ProCrnRelCrState and has the values: • PRO_CRN_REL_CR_IN_FORMED—Creates corner relief geometry only when the corner bends are in formed state. • PRO_CRN_REL_CR_IN_UNBEND_ONLY—Creates corner relief geometry only when the corner bends are in unbend state.
PRO_E_CORNER_RELIEFS_ARR	Array	An array element of corner relief sets.

Element ID	Data Type	Description
PRO_E_CORNER_RELIEF_SET	Compound	This compound element contains a corner relief set.
PRO_E_CORNER_RELIEF_REF_ARR	Array	An array element that contains the corners selected in this set.
PRO_E_CORNER_RELIEF_REF_SET	Compound	A set of elements that defines a selected corner.
PRO_E_CORNER_RELIEF_REF_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of corner relief using the enumerated data type ProCrnRelRefType. The valid values for this element are: • PRO_CRN_REL_3_SURFACES—Corner relief must be applied to the specified corners in the model. • PRO_CRN_REL_ALL—Corner relief must be applied to all the corners in the model.
PRO_E_CORNER_RELIEF_REF_FLAT	PRO_VALUE_TYPE_SELECTION	Specifies selection of a flat surface that is used to locate the corner.
PRO_E_CORNER_RELIEF_REF_BND_1	PRO_VALUE_TYPE_SELECTION	Specifies selection of a first cylindrical surface that is used to locate the corner.
PRO_E_CORNER_RELIEF_REF_BND_2	PRO_VALUE_TYPE_SELECTION	Specifies selection of a second cylindrical surface that is used to locate the corner.
PRO_E_CORNER_RELIEF_DEFINE	Compound	A corner relief compound element. Refer to the section Corner Relief Options on page 1380 for more details on this element and subsequent child elements.

Corner Relief Options

The compound element PRO_E_CORNER_RELIEF_ DEFINE defines the options and values for a corner relief feature.

Element Tree for Corner Relief Feature Options

```
|-- PRO_E_CORNER_RELIEF_DEFINE (compound)
    |
 |-- PRO_E_SMT_CORNER_RELIEF (compound)
     |
     |-- PRO_E_SMT_CORNER_RELIEF_TYPE (option)
     |
     |-- PRO_E_SMT_CORNER_RELIEF_WIDTH (compound)
     |   |
     |      |-- PRO_E_SMT_CORNER_RELIEF_WIDTH_TYPE (option)
     |   |
     |   |-- PRO_E_SMT_CORNER_RELIEF_WIDTH_VAL (double)
     |
     |-- PRO_E_SMT_CORNER_RELIEF_DEPTH (compound)
     |   |
     |   |-- PRO_E_SMT_CORNER_RELIEF_DEPTH_TYPE (option)
     |      |
     |      |-- PRO_E_SMT_CORNER_RELIEF_DEPTH_VAL (double)
     |
     |-- PRO_E_SMT_CORNER_RELIEF_ROTATE (compound)
     |      |
     |   |-- PRO_E_SMT_CORNER_RELIEF_ROTATE_ADD (option)
     |   |
     |   |-- PRO_E_SMT_CORNER_RELIEF_ROTATE_VAL (double)
     |
     |-- PRO_E_SMT_CORNER_RELIEF_OFFSET (compound)
     |      |
     |   |-- PRO_E_SMT_CORNER_RELIEF_OFFSET_ADD (option)
     |      |
     |      |-- PRO_E_SMT_CORNER_RELIEF_OFFSET_VAL (double)
     |
       |-- PRO_E_CORNER_RELIEFS_DIM_SCHEME (option)
```

The elements of PRO_E_CORNER_RELIEF_ DEFINE are described as follows:

Element ID	Data Type	Description
PRO_E_CORNER_RELIEF_ DEFINE	Compound	A corner relief compound element.
PRO_E_SMT_CORNER_ RELIEF	Compound	This compound element defines the corner relief properties.
PRO_E_SMT_CORNER_ RELIEF_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of corner relief using the enumerated data type ProCornerRelType and has the following values:

Element ID	Data Type	Description
		• PRO_CORNER_RELIEF_V_NOTCH—Adds a V notch shape cut at the corners. • PRO_CORNER_RELIEF_CIRCULAR—Adds a circular shape relief at the corners with a radius dimension. • PRO_CORNER_RELIEF_RECTANGULAR—Adds a rectangular relief at the corners with a specified width and depth. • PRO_CORNER_RELIEF_OBROUND—Adds an obround relief at the corners with a specified diameter and depth. • PRO_CORNER_RELIEF_NO—Does not add relief and generates square corners. • PRO_CORNER_RELIEF_PARAM—Adds the corner relief as set by the SMT_DFLT_CRNR_REL_TYPE parameter in Creo Parametric.
PRO_E_SMT_CORNER_RELIEF_WIDTH	Compound	This compound element defines the width type and width value for corner relief.
PRO_E_SMT_CORNER_RELIEF_WIDTH_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of width for the corner relief using the enumerated data type ProSmdRelType and uses one of the values listed in Relation Value Types on page 1342
PRO_E_SMT_CORNER_RELIEF_WIDTH_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of width for the corner relief.
PRO_E_SMT_CORNER_RELIEF_DEPTH	Compound	This compound element defines the depth type and depth value for corner relief.
PRO_E_SMT_CORNER_RELIEF_DEPTH_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of depth for the corner relief using the enumerated data type ProCornerRlfDepthType and uses one of the values listed in

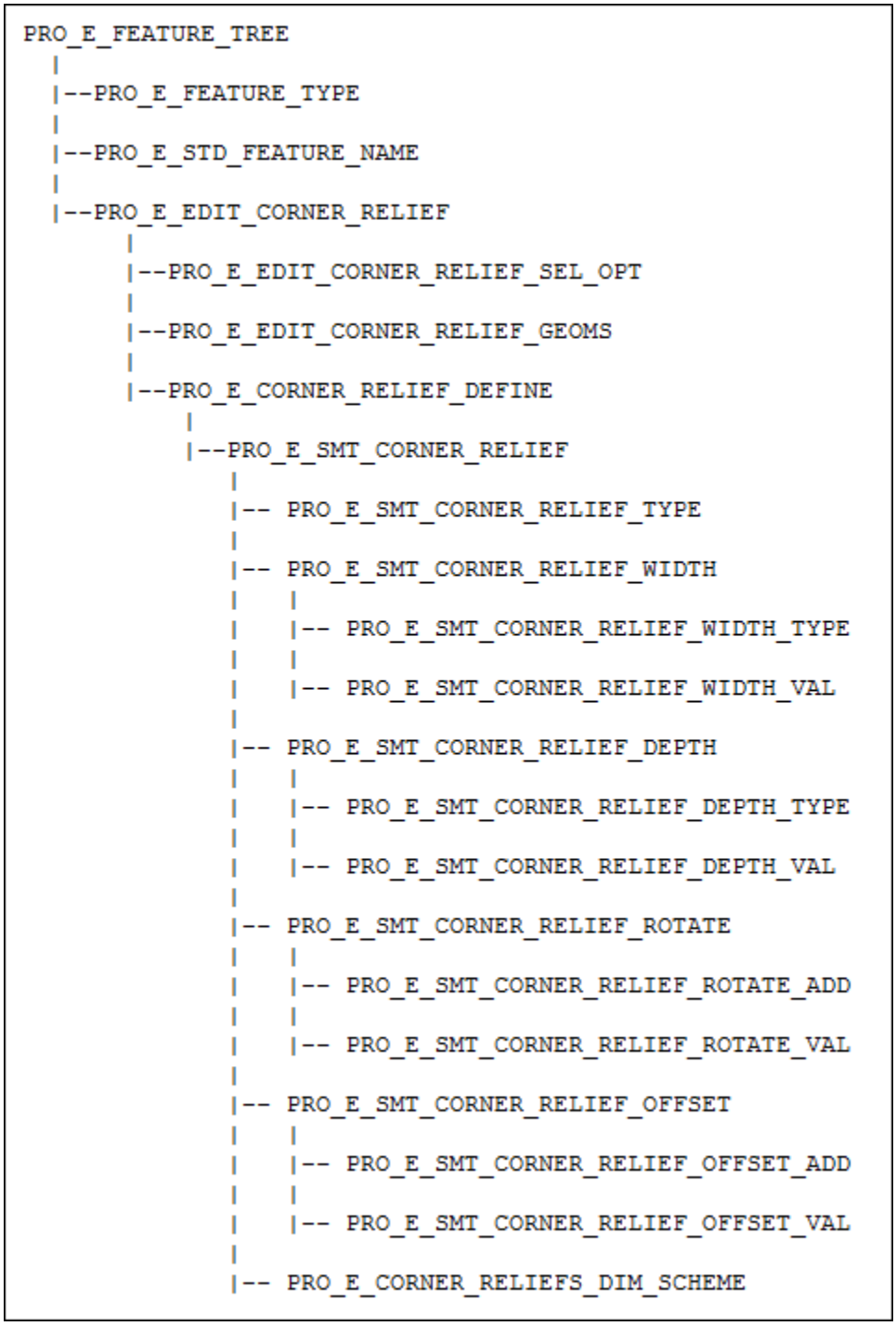
Element ID	Data Type	Description
		Relation Value Types on page 1342
PRO_E_SMT_CORNER_RELIEF_DEPTH_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of depth value for the corner relief.
PRO_E_SMT_CORNER_RELIEF_ROTATE	Compound	This compound element defines the rotation parameters for corner relief and is applicable only if PRO_E_SMT_CORNER_RELIEF_TYPE value is PRO_CORNER_RELIEF_OBROUND or PRO_CORNER_RELIEF_RECTANGULAR.
PRO_E_SMT_CORNER_RELIEF_ROTATE_ADD	PRO_VALUE_TYPE_INT	Specifies if rotation should be added to the relief placement. The valid values for this element are: • PRO_B_TRUE • PRO_B_FALSE
PRO_E_SMT_CORNER_RELIEF_ROTATE_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of rotation.
PRO_E_SMT_CORNER_RELIEF_OFFSET	Compound	This compound element defines the offset parameters for corner relief and is applicable only if PRO_E_SMT_CORNER_RELIEF_TYPE value is PRO_CORNER_RELIEF_CIRCULAR, PRO_CORNER_RELIEF_OBROUND or PRO_CORNER_RELIEF_RECTANGULAR.
PRO_E_SMT_CORNER_RELIEF_OFFSET_ADD	PRO_VALUE_TYPE_INT	Specifies if offset should be added to the relief placement. The valid values for this element are: • PRO_B_TRUE • PRO_B_FALSE

Element ID	Data Type	Description
PRO_E_SMT_CORNER_RELIEF_OFFSET_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the offset value.
PRO_E_CORNER_RELIEFS_DIM_SCHEME	PRO_VALUE_TYPE_INT	Specifies the dimension scheme using the enumerated data type ProCrnRelDimRefType and has the values: • PRO_CRN_REL_DIM_REF_BEND_AXES_XSECTION—Places the relief at the point where the bend edges intersect. • PRO_CRN_REL_DIM_REF_CORNER_VERTEX—Places the relief at the point where the bend lines intersect.

Editing Corner Relief Feature

The **Edit Corner Relief** feature removes or edits multiple corner relief design objects in your model. When you edit corner reliefs, you can change the width and depth of different types of reliefs and you can set bounding surfaces to remove. You can edit corner reliefs to no relief or to any one of the types of corner reliefs.

The element tree for the **Edit Corner Relief** feature is documented in the header file `ProSmtEditCornerRelief.h` and is shown in the following figure:



The elements in this tree are described as follows:

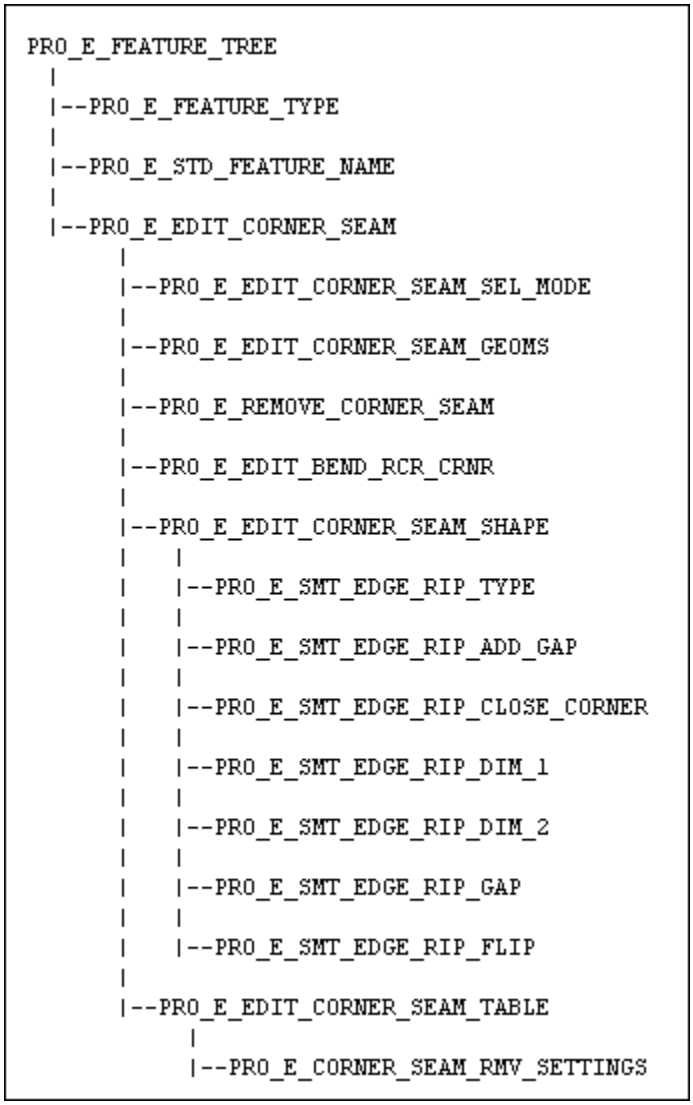
Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_EDIT_CORNER_RELIEF.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Edit_Corner_Relief_1.
PRO_E_EDIT_CORNER_RELIEF	Compound	This compound element defines the options and sets the values for editing corner reliefs.
PRO_E_EDIT_CORNER_RELIEF_SEL_OPT	PRO_VALUE_TYPE_INT	Specifies the corner relief state using the enumerated data type ProCrnRelCrState. It has the following valid values: • PRO_CRN_REL_CR_IN_FORMED—Creates corner relief geometry only when the corner bends are in formed state. • PRO_CRN_REL_CR_IN_UNBEND_ONLY—Creates corner relief geometry only when the corner bends are in unbend state.
PRO_E_CORNER_RELIEF_DEFINE	Compound	A corner relief compound element. Refer to the section Corner Relief Options on page 1380 for more details on this element and subsequent child elements.

Editing Corner Seams

The Edit Corner Seam feature enables you to remove or edit multiple corner seam design objects in your model.

The element tree for the Edit Corner Seam feature is documented in the header file ProSmtEditCornerSeam.h and is as shown in the following figure:

Feature Element Tree for Edit Corner Seam Feature



The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_EDIT_CORNER_SEAM.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Mandatory element. Specifies the name of the feature. The default value is Edit_Corner_Seam_1.
PRO_E_EDIT_CORNER_SEAM	Compound	Mandatory element. This compound element defines the options and sets the values for editing corner seams.

Element ID	Data Type	Description
PRO_E_EDIT_CORNER_SEAM_SEL_MODE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the mode for selecting corner seams. The valid values are: • PRO_SMT_RECOGNITION_MANUAL_SEL—Specifies manual selection of corner seams to edit. • PRO_SMT_RECOGNITION_AUTO_SEL—Specifies automatic selection of corner seams to edit.
PRO_E_EDIT_CORNER_SEAM_GEOMS	Multi Collector	This element is mandatory when the selection mode is set to PRO_SMT_RECOGNITION_MANUAL_SEL. Specifies the selection of geometry for corner seams.
PRO_E_REMOVE_CORNER_SEAM	PRO_VALUE_TYPE_BOOLEAN	Specifies that the corner seam must be removed.
PRO_E_EDIT_CORNER_SEAM_SHAPE	Compound	This element is mandatory when the element PRO_E_REMOVE_CORNER_SEAM is set to false. This compound element defines the options and sets the values for corner seam edges.
PRO_E_EDIT_BEND_RCR_CRNR	PRO_VALUE_BOOLEAN	Specifies if the corner reliefs must be automatically changed to V notch corner type.
PRO_E_CRNR_SEAM_CR_RND_CHMF	PRO_VALUE_TYPE_INT	Mandatory element. Specifies if the rounds and chamfers must be recreated after the corner seams are edited. The input to the element are the values defined by the enumerated data type ProEditBendCrRndChmfOpt. The valid values are: • PRO_ED_CR_CRNR_SEAM_RND_CHMF— The rounds and chamfers are recreated. • PRO_ED_NO_CR_CRNR_SEAM_RND_CHMF— The rounds and chamfers are not recreated.
PRO_E_SMT_EDGE_RIP_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of corner seam using the enumerated data type ProEdgeRipType. The valid values are:

Element ID	Data Type	Description
		• PRO_EDGE_RIP_OPEN—Edits the corner seam at the intersection point of edges. • PRO_EDGE_RIP_BLIND—Edits the corner seam with a gap specified by two dimensions. • PRO_EDGE_RIP_MITER_CUT—Edits the corner seam with a gap specified by a single dimension. • PRO_EDGE_RIP_OVERLAP—Edits the corner seam such that one edge overlaps the other.
PRO_E_SMT_EDGE_RIP_ADD_GAP	PRO_VALUE_TYPE_BOOLEAN	Specifies whether to add a gap clearance. This element is applicable only if the element PRO_E_SMT_EDGE_RIP_TYPE is set to PRO_EDGE_RIP_MITER_CUT or PRO_EDGE_RIP_OVERLAP.
PRO_E_SMT_EDGE_RIP_CLOSE_CORNER	PRO_VALUE_TYPE_BOOLEAN	Specifies if the gap between the bend surfaces of a corner relief must be closed. This element is applicable only if the element PRO_E_SMT_EDGE_RIP_TYPE is set to PRO_EDGE_RIP_OPEN.
PRO_E_SMT_EDGE_RIP_DIM_1	Compound	This element is mandatory when the element PRO_E_SMT_EDGE_RIP_TYPE is set to PRO_EDGE_RIP_BLIND. Specifies a compound element that defines the properties of side 1.
PRO_E_SMT_DIMENSION_TYPE	PRO_VALUE_TYPE_INT	Specifies the dimension type. For PRO_EDGE_RIP_DIM_TYPE_BLIND dimension type, the relation value is set to PRO_DIM_ENTER.
PRO_E_SMT_DIMENSION_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the dimension value for side 1.
PRO_E_SMT_EDGE_RIP_DIM_2	Compound	This element is mandatory when the element PRO_E_SMT_EDGE_RIP_TYPE is set to PRO_EDGE_RIP_BLIND. Specifies a compound element that defines the properties of side 2.
PRO_E_SMT_DIMENSION_TYPE	PRO_VALUE_TYPE_INT	Specifies the dimension type. For PRO_EDGE_RIP_DIM_TYPE_BLIND dimension type, the

Element ID	Data Type	Description
		relation value is set to PRO_ DIM_ENTER.
PRO_E_SMT_DIMENSION_ VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the dimension value for side 2.
PRO_E_SMT_EDGE_RIP_GAP	Compound	This element is mandatory when the element PRO_E_SMT_EDGE_ RIP_TYPE is set to PRO_EDGE_ RIP_MITER_CUT or PRO_E_ SMT_EDGE_RIP_ADD_GAP is set to true. Specifies the properties for the gap.
PRO_E_SMT_DIMENSION_ TYPE	PRO_VALUE_TYPE_INT	Specifies the gap type. For PRO_ EDGE_RIP_MITER_CUT and PRO_EDGE_RIP_OVERLAP dimension types the relation value is set to PRO_ DIM_ENTER.
PRO_E_SMT_DIMENSION_ VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the value for the gap.
PRO_E_SMT_EDGE_RIP_FLIP	PRO_VALUE_TYPE_BOOLEAN	Specifies if the side of corner seam that overlaps must be flipped.
PRO_E_EDIT_CORNER_SEAM_ TABLE	Array	Specifies an array element that defines the options to remove the corner seam.
PRO_E_CORNER_SEAM_RMV_ SETTINGS	Compound	Specifies a compound element that defines the options to remove the corner seam.
PRO_E_EDIT_CORNER_SEAM_ RMV_REFS	Compound	Specifies a compound element that defines the corner reference geometry to which the seam is attached. The surface should be planar or two cylinders.
PRO_E_CORNER_SEAM_RMV_ FLAT	PRO_VALUE_TYPE_ SELECTION	Specifies a flat surface.
PRO_E_CORNER_SEAM_RMV_ BEND_1	PRO_VALUE_TYPE_ SELECTION	Specifies the first bend surface.
PRO_E_CORNER_SEAM_RMV_ BEND_2	PRO_VALUE_TYPE_ SELECTION	Specifies the second bend surface.
PRO_E_CORNER_SEAM_ BOUNDARIES	Multi Collector	Specifies collection of bounding surfaces.
PRO_E_CORNER_SEAM_RMV_ SIDE_1	Compound	Specifies a compound element that defines the properties of side 1.
PRO_E_CORNER_SEAM_RMV_ DEFAULT	PRO_VALUE_TYPE_BOOLEAN	Specifies the default option to remove the corner seam for side 1.
PRO_E_EDIT_CORNER_SEM_ RMV_METHOD	PRO_VALUE_TYPE_INT	Specifies the method to remove the corner seam for side 1. The valid values are specified using the enumerated data type ProEditCornerSeamRemove Type:

Element ID	Data Type	Description
		• PRO_CORNER_SEAM_REMOVE_TANGENT—Extends or trims the bounding surface making it a planar surface tangent to original surface. • PRO_CORNER_SEAM_REMOVE_SAME—Extends or trims the bounding surface by continuing past its original boundaries and keeping the same type of surface. • PRO_CORNER_SEAM_REMOVE_PARALLEL—Extends or trims the bounding surface parallel to the bend axis. • PRO_CORNER_SEAM_REMOVE_COMMON_VERTEX—Extends or trims the bounding surface normal to the corner. • PRO_CORNER_SEAM_REMOVE_NORMAL—Finds the common vertex for intersection of both bounding surfaces.
PRO_E_CORNER_SEAM_RMV_SIDE_2	Compound	Specifies a compound element that defines the properties of side 2.
PRO_E_CORNER_SEAM_RMV_DEFAULT	PRO_VALUE_TYPE_BOOLEAN	Specifies the default option to remove the corner seam for side 2.
PRO_E_EDIT_CORNER_SEM_RMV_METHOD	PRO_VALUE_TYPE_INT	Specifies the method to remove the corner seam for side 2 using the enumerated data type ProEditCornerSeamRemoveType.

Bend Feature

The Bend feature allows you to bend the sheet metal in different ways using the bend line or an edge or a curve and by defining specific radius and angle.

The element tree for the Bend feature is documented in the header file ProSmtBend.h and is as shown in the following figure:

Feature Element Tree for Bend Feature

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_SMT_BEND_FORM
   |
   |--PRO_E_SMT_BEND_LINE
   |
   |--PRO_E_SMT_BEND_FIXED_SIDE
   |
   |--PRO_E_SMT_BEND_LOCATION
   |
   |--PRO_E_SMT_BEND_DIRECTION
   |
   |--PRO_E_SMT_BEND_ANGLE
   |     |
   |     |--PRO_E_SMT_BEND_ANGLE_TYPE
   |     |
   |     |--PRO_E_SMT_BEND_ANGLE_VALUE
   |
   |--PRO_E_SMT_BEND_TRANS_FLIP
   |
   |--PRO_E_SMT_BEND_TRANS_AREAS
   |     |
   |     |--PRO_E_SMT_BEND_TRANS_SET
   |           |
   |           |--PRO_E_STD_SECTION
   |
   |--PRO_E_SMT_FILLETS
   |     |
   |     |--PRO_E_SMT_FILLETS_SIDE
   |     |
   |     |--PRO_E_SMT_FILLETS_VALUE
   |
   |--PRO_E_SMT_BEND_RELIEF
   |
   |--PRO_E_SMT_DEV_LEN_CALCULATION
   |     |
   |     |--PRO_E_SMT_DEV_LEN_SOURCE
   |     |
   |     |--PRO_E_SMT_DEV_LEN_Y_FACTOR
   |     |     |
   |     |     |--PRO_E_SMT_DEV_LEN_Y_FACTOR_TYPE
   |     |     |
   |     |     |--PRO_E_SMT_DEV_LEN_Y_FACTOR_VALUE
   |     |
   |     |--PRO_E_SMT_DEV_LEN_BEND_TABLE
   |
```

The elements in this tree are described as follows:

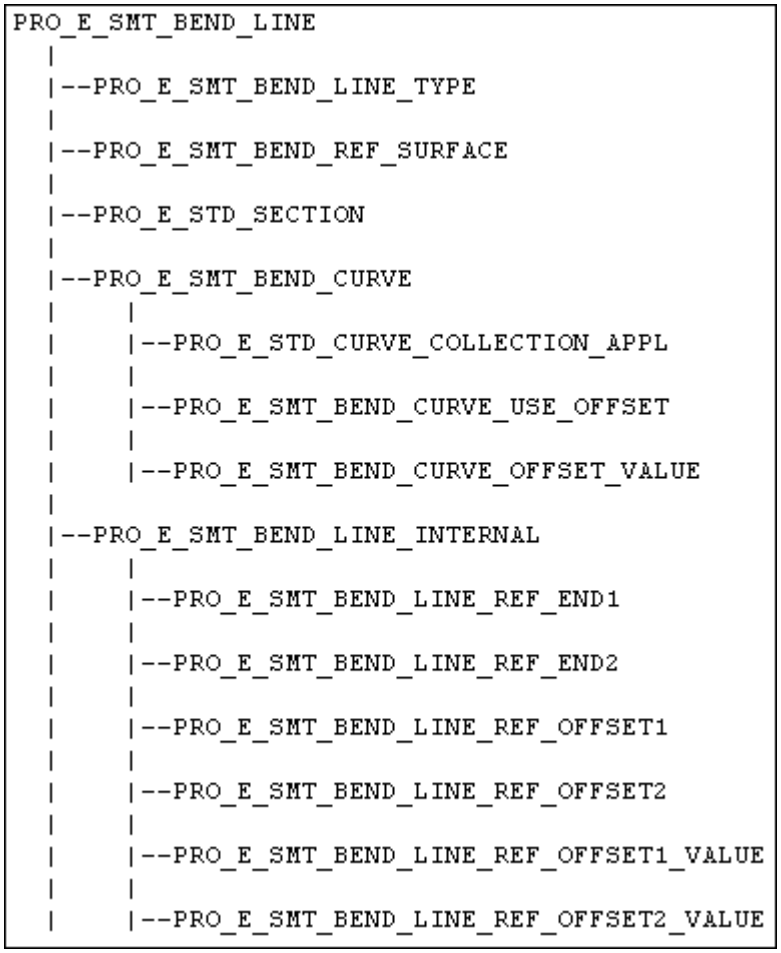
Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_BEND.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Bend_1.
PRO_E_SMT_BEND_FORM	PRO_VALUE_TYPE_INT	Specifies the type of bend using the enumerated data type ProBendForm. The valid values for this element are: • PRO_SMT_BEND_FORM_ANGLE—Specifies an angled type of bend. • PRO_SMT_BEND_FORM_ROLL—Specifies a rolled type of bend.
PRO_E_SMT_BEND_LINE	Compound	This compound element defines the bend line properties. For more information, see the section Bend Line Elements on page 1394.
PRO_E_SMT_BEND_FIXED_SIDE	PRO_VALUE_TYPE_INT	Specifies the fixed side of the bend using the enumerated data type ProBendSide. The valid values for this element are: • PRO_SMT_BEND_SIDE_ONE—Specifies first side as the fixed side. • PRO_SMT_BEND_SIDE_TWO—Specifies the second side as the fixed side.
PRO_E_SMT_BEND_LOCATION	PRO_VALUE_TYPE_INT	Specifies the location of the bend in relation to the bend line, using the enumerated data type ProBendSide. The valid values for this element are: • PRO_SMT_BEND_SIDE_ONE—Specifies a bend up to the bend line. • PRO_SMT_BEND_SIDE_TWO—Specifies a bend up to the other side of the bend line. • PRO_SMT_BEND_BOTH_SIDES—Specifies a bend centered on both sides of the bend line.
PRO_E_SMT_BEND_DIRECTION	PRO_VALUE_TYPE_INT	Specifies the direction of the bend using the enumerated data type ProBendSide. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_SMT_BEND_SIDE_ONE—Specifies the bend direction normal to the selected surface. • PRO_SMT_BEND_SIDE_TWO—Flips the bend direction.
PRO_E_SMT_BEND_ANGLE	Compound	This compound element and its sub-elements are available when the element PRO_E_SMT_BEND_FORM has its value as PRO_SMT_BEND_FORM_ANGLE. The compound element defines the bend angle.
PRO_E_SMT_BEND_ANGLE_TYPE	PRO_VALUE_TYPE_INT	Specifies the bend angle type using the enumerated data type ProBendAngleType. The valid values for this element are: • PRO_SMT_BEND_ANGLE_INTERNAL—Specifies the resulting internal bend angle. • PRO_SMT_BEND_ANGLE_EXTERNAL—Specifies the bend angle deflection from straight.
PRO_E_SMT_BEND_ANGLE_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the bend angle.
PRO_E_SMT_BEND_TRANS_FLIP	PRO_VALUE_TYPE_INT	Flips the direction where the bend cylinder will be created along the bend transition line using the enumerated data type ProBendSide. The valid values for this element are: • PRO_SMT_BEND_SIDE_ONE— Creates the cylinder along the first line of the transition. • PRO_SMT_BEND_SIDE_TWO—Creates the cylinder along the second line of the transition. **Note** The transition flip element is available only if there is one transition set in the feature.
PRO_E_SMT_BEND_TRANS_AREAS	Array	An array element of bend transition lines that defines the bend transition area.
PRO_E_SMT_BEND_TRANS_SET	Compound	This compound element defines the bend transition lines.

Element ID	Data Type	Description
PRO_E_STD_SECTION	Compound	This compound element specifies a sketched section for the bend line. For more information on how to create features that contain sketched sections, refer to the section Creating Features Containing Sections on page 1006.
PRO_E_SMT_FILLETS	Compound	This compound element defines the bend properties of the sheet metal wall and the value of bend radius.
PRO_E_SMT_FILLETS_SIDE	PRO_VALUE_TYPE_INT	Specifies the fillet side. The valid values are: • PRO_BEND_RAD_OUTSIDE—Applies the bend radius to the outer surface of the bend. • PRO_BEND_RAD_INSIDE—Applies the bend radius to the inner surface of the bend. • PRO_BEND_RAD_PARAMETER—Applies the bend radius at the dimension location set by the SMT_DFLT_RADIUS_SIDE parameter in Creo Parametric.
PRO_E_SMT_FILLETS_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the bend radius.
PRO_E_SMT_BEND_RELIEF	Compound	This compound element defines the bend relief at the edges. For more information see the section Bend Relief Elements on page 1398.
PRO_E_SMT_DEV_LEN_CALCULATION	Compound	This compound element defines the method used to calculate the Developed Length dimensions for the bends. For more information see the section The Element Subtree for Length Calculation on page 1327

Bend Line Elements

This compound element PRO_E_SMT_BEND_LINE defines the bend line, its properties and references for sheet metal bends.



The elements of `PRO_E_SMT_BEND_LINE` are described as follows:

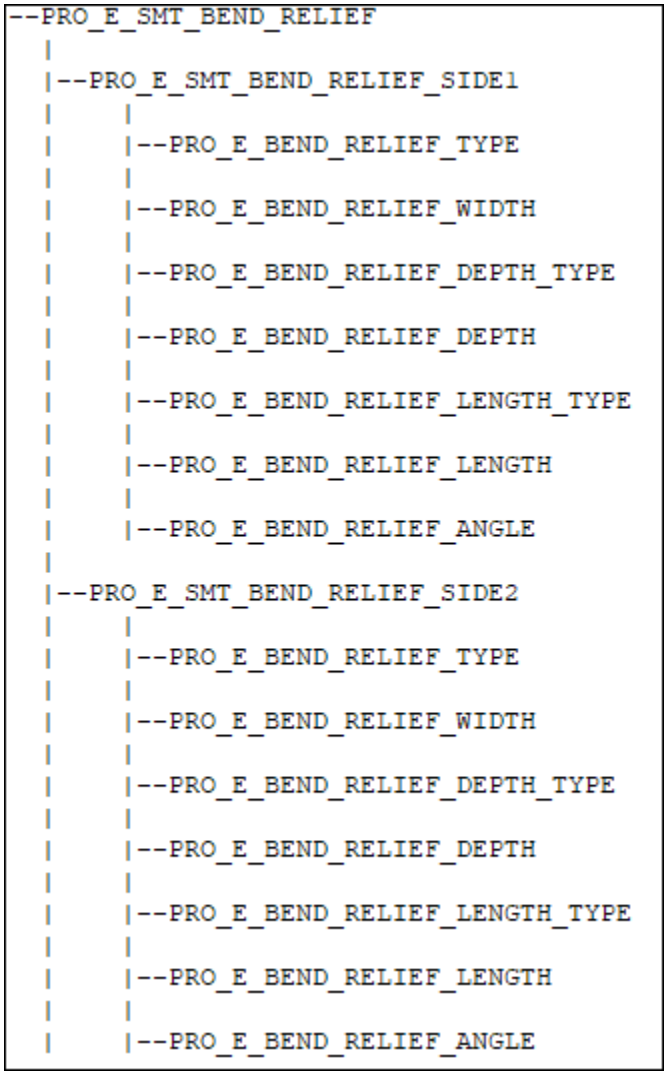
Element ID	Data Type	Description
PRO_E_SMT_BEND_LINE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of bend using the enumerated data type ProBendLineType. The valid values for this element are: • PRO_SMT_BEND_LINE_NOT_DEFINED—Specifies an undefined bend line. • PRO_SMT_BEND_LINE_SKETCH Specifies a user defined bend line. • PRO_SMT_BEND_LINE_CURVE—Specifies a linear chain to be used as the bend line. • PRO_SMT_BEND_LINE_INTERNAL_LINE—Specifies the bend line that will be created based on the compound element PRO_E_SMT_BEND_LINE_INTERNAL.
PRO_E_SMT_BEND_REF_SURFACE	PRO_VALUE_TYPE_SELECTION	This element is available when the value of element PRO_E_SMT_BEND_LINE_TYPE is PRO_SMT_BEND_LINE_INTERNAL_LINE or PRO_SMT_BEND_LINE_SKETCH. Specifies the surface to be bent. This is the surface on which the bend line is set.
PRO_E_STD_SECTION	Compound	This compound element and its sub-elements are available only if the element PRO_E_SMT_BEND_LINE_TYPE has its value as PRO_SMT_BEND_LINE_SKETCH. The compound element specifies a sketched section for the bend line. For more information on how to create features that contain sketched sections, refer to the section Creating Features Containing Sections on page 1147.
PRO_E_SMT_BEND_CURVE	Compound	This element and its sub-elements are available when the element PRO_E_SMT_BEND_LINE_TYPE has its value as PRO_SMT_BEND_LINE_CURVE.

Element ID	Data Type	Description
		This compound element defines the curve references and bend curve offset value.
PRO_E_STD_CURVE_ COLLECTION_APPL	PRO_VALUE_TYPE_POINTER	Specifies a surface edge or a curve that defines the bend line.
PRO_E_SMT_BEND_CURVE_ USE_OFFSET	PRO_VALUE_TYPE_INT	Specifies whether to offset the bend curve from the selected reference. The valid values for this element are: • PRO_B_TRUE • PRO_B_FALSE
PRO_E_SMT_BEND_CURVE _OFFSET_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the offset value for the bend curve.
PRO_E_SMT_BEND_LINE_ INTERNAL	Compound	This element and its sub-elements are available when the element PRO_E_SMT_BEND_LINE_ TYPE has its value as PRO_SMT_ BEND_LINE_INTERNAL_LINE. The compound element defines the bend line reference ends, offsets and offset values.
PRO_E_SMT_BEND_LINE_REF _END1	PRO_VALUE_TYPE_ SELECTION	Specifies a vertex or an edge as the placement reference for the first end of the bend line.
PRO_E_SMT_BEND_LINE_REF _END2	PRO_VALUE_TYPE_ SELECTION	Specifies a vertex or an edge as the placement reference for the second end of the bend line.
PRO_E_SMT_BEND_LINE_REF _OFFSET1	PRO_VALUE_TYPE_ SELECTION	This element is available when you choose an edge reference in the element PRO_E_SMT_BEND_ LINE_ REF_END1. Specifies an edge as the offset reference for the first end of the bend line.
PRO_E_SMT_BEND_LINE_REF _OFFSET2	PRO_VALUE_TYPE_ SELECTION	This element is available when you choose an edge reference in the element PRO_E_SMT_BEND_ LINE_ REF_END2. Specifies an edge as the offset reference for the second end of the bend line.

Element ID	Data Type	Description
PRO_E_SMT_BEND_LINE_REF_OFFSET1_VALUE	PRO_VALUE_TYPE_DOUBLE	This element is available when you choose an edge reference in the element PRO_E_SMT_BEND_LINE_REF_END1. Specifies the offset value from the first end of the bend line.
PRO_E_SMT_BEND_LINE_REF_OFFSET2_VALUE	PRO_VALUE_TYPE_DOUBLE	This element is available when you choose an edge reference in the element PRO_E_SMT_BEND_LINE_REF_END2. Specifies the offset value from the second end of the bend line.

Bend Relief Elements

The compound element PRO_E_SMT_BEND_RELIEF defines the bend relief elements. The relief can be specified differently on each side of the bend.



The two main elements of `PRO_E_SMT_BEND_RELIEF` are:

- `PRO_E_SMT_BEND_RELIEF_SIDE1`—This compound element specifies the bend relief applied to the first side of the end of the bend.
- `PRO_E_SMT_BEND_RELIEF_SIDE2`—This compound element specifies the bend relief applied to the second side of the end of the bend.

The following elements are common to the both the compound elements:

Element ID	Data Type	Description
PRO_E_BEND_RELIEF_TYPE	PRO_VALUE_TYPE_INT	Specifies the relief type using the enumerated data type ProBendRlfType. The valid values for this element are: • PRO_BEND_RLF_NONE— Specifies no relief. • PRO_BEND_RLF_RIP— Specifies ripping of the material. • PRO_BEND_RLF_STRETCH— Specifies stretching of the material for bend relief. • PRO_BEND_RLF_RECTANGULAR— Specifies adding a rectangular relief. • PRO_BEND_RLF_OBROUND— Specifies adding an obround relief. • PRO_BEND_RLF_PARAM— Specifies relief type set by the part parameter SMT_DFLT_BEND_REL_TYPE.
PRO_E_BEND_RELIEF_WIDTH	PRO_VALUE_TYPE_DOUBLE	Specifies the relief width and is applicable only if the value of PRO_E_BEND_RELIEF_TYPE is PRO_BEND_RLF_RECTANGULAR, PRO_BEND_RLF_STRETCH or PRO_BEND_RLF_OBROUND.
PRO_E_BEND_RELIEF_DEPTH_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of depth relief. The valid values for this element are defined in the enumerated type ProBendRlfDepthType and are as follows: • PRO_BEND_RLF_DEPTH_NOT_USED— Specifies that the depth for the relief is not used. • PRO_BEND_RLF_DEPTH_BLIND— Creates a relief through the geometry as per the specified value. • PRO_BEND_RLF_DEPTH_UP_TO_BEND— Creates a relief up to the bend. • PRO_BEND_RLF_DEPTH_TAN_TO_BEND— Creates a relief tangential to the bend. PRO_BEND_RLF_DEPTH_TAN_TO_BEND is applicable

Element ID	Data Type	Description
		only when relief type is set to PRO_BEND_RLF _OBROUND.

Element ID	Data Type	Description
		• PRO_BEND_RLF_DEPTH_TYPE_PARAM— Specifies the depth type of the relief at the dimension location and is set by part parameter SMT_DFLT_BEND_REL_DEPTH_TYPE. This element decides the visibility of the bend relief depth element PRO_E_BEND_RELIEF_DEPTH. If PRO_E_BEND_RELIEF_DEPTH_TYPE is set to PRO_BEND_RLF_DEPTH_BLIND or PRO_BEND_RLF_DEPTH_TYPE_PARAM and the part parameter SMT_DFLT_BEND_REL_DEPTH_TYPE is set to **Blind**, then the existing element PRO_E_BEND_RELIEF_DEPTH is used. PTC recommends that you define the element PRO_E_BEND_RELIEF_DEPTH_TYPE explicitly for all Creo Parametric TOOLKIT applications. If the element PRO_E_BEND_RELIEF_DEPTH_TYPE is not defined, the default value is used. The default value from Creo Parametric 1.0 onwards, depends on the part parameter SMT_DFLT_BEND_REL_DEPTH_TYPE and the configuration option SMT_DRIVE_TOOLS_BY_PARAMETERS. If the value of the configuration option SMT_DRIVE_TOOLS_BY_PARAMETERS is set to No, then the default value is the last bend relief type, as selected in Creo Parametric during the creation of the new feature. For the Pro/TOOLKIT applications prior to Creo Parametric 1.0, if the default value is not Blind, then the

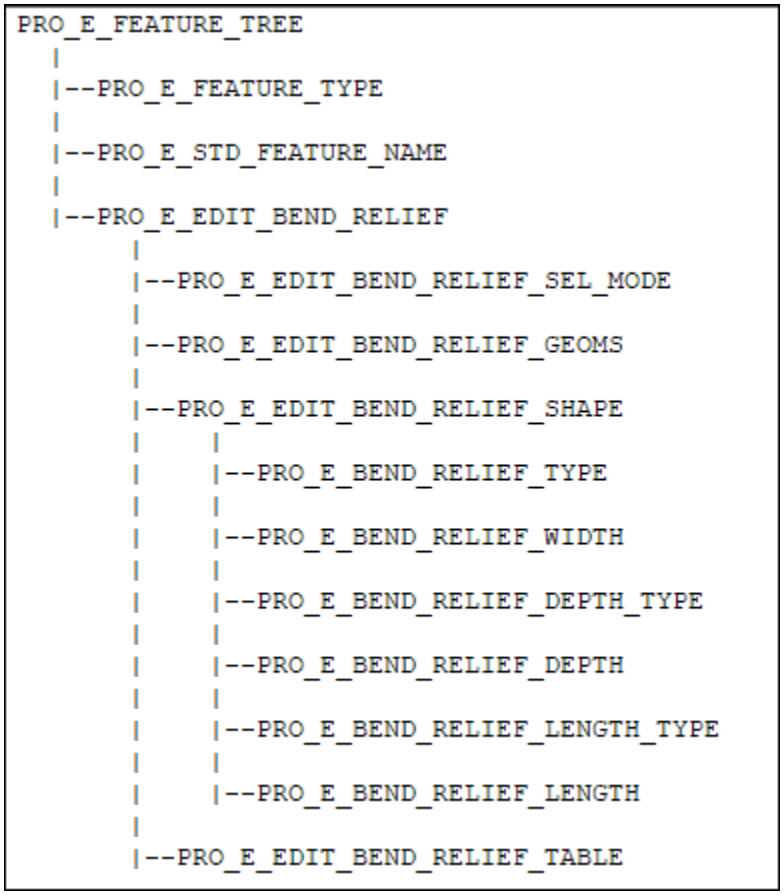
Element ID	Data Type	Description
		element PRO_E_BEND_RELIEF_DEPTH_TYPE is not used. For such cases, set the PRO_E_BEND_RELIEF_DEPTH_TYPE to PRO_BEND_RLF_DEPTH_BLIND.
PRO_E_BEND_RELIEF_DEPTH	PRO_VALUE_TYPE_DOUBLE	Specifies the relief depth and is applicable only if the value of PRO_E_BEND_RELIEF_TYPE is PRO_BEND_RLF_RECTANGULAR or PRO_BEND_RLF_OBROUND.
PRO_E_BEND_RELIEF_LENGTH_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the relief length and is defined by the enumerated data type ProBendRlfLengthType. The valid values follow: • PRO_BEND_RLF_LENGTH_NOT_USED • PRO_BEND_RLF_LENGTH_BLIND—Creates the bend reliefs with a length of the specified value. • PRO_BEND_RLF_LENGTH_TO_NEXT—Creates the bend reliefs with a length to the next surface. • PRO_BEND_RLF_LENGTH_THROUGH_ALL—Creates the bend reliefs through all surfaces. • PRO_BEND_RLF_LENGTH_TYPE_PARAM—Uses the SMT_DFLT_BEND_REL_LENGTH_TYPE parameter value.
PRO_E_BEND_RELIEF_LENGTH	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the relief length.
PRO_E_BEND_RELIEF_ANGLE	PRO_VALUE_TYPE_DOUBLE	Specifies the relief angle and is applicable only if the value of PRO_E_BEND_RELIEF_TYPE is PRO_BEND_RLF_STRETCH.

Editing Bend Reliefs

The Edit Bend Relief feature enables you to edit bend reliefs in existing bends.

The element tree for the Edit Bend Relief feature is documented in the header file ProSmtEditBendRelief.h and is shown in the following figure:

Feature Element Tree for Edit Bend Relief Feature



The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of feature. The value of this feature must be PRO_FEAT_EDIT_BEND_RELIEF.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Mandatory element. Specifies the name of the feature. The default value is EDIT_BEND_RELIEF_1.
PPRO_E_EDIT_BEND_RELIEF	Compound	Mandatory element. This compound element defines the options and sets the values for editing a bend relief.
PRO_E_EDIT_BEND_RELIEF_SEL_MODE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the mode for selecting bend reliefs. The valid values are:

Element ID	Data Type	Description
		• PRO_SMT_RECOGNITION_MANUAL_SEL—Specifies manual selection of bend reliefs to edit. • PRO_SMT_RECOGNITION_AUTO_SEL—Specifies automatic selection of bend reliefs to edit.
PRO_E_EDIT_BEND_RELIEF_GEOMS	Multi Collector	This element is mandatory when the selection mode is set to PRO_SMT_RECOGNITION_MANUAL_SEL. Specifies the selection of geometry for bend reliefs. You can select valid bend relief design objects, geometry of recognizable bend reliefs, or surfaces of bends for which bend reliefs can be edited. You can also select planes with adjacent bends. If an intent surface is selected, all the surfaces which are not relevant will be ignored. Thickness edges can be selected when side surfaces cannot be selected. Bend relief vertices can be selected when neither side surfaces nor thickness edges can be selected.
PRO_E_EDIT_BEND_RELIEF_SHAPE	Compound	Mandatory element. This compound element specifies the options that define the shape of a bend relief.
PRO_E_BEND_RELIEF_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of bend relief. The valid values are: • PRO_BEND_RLF_RIP—The selected bend reliefs are edited to rip reliefs with no dimensions. • PRO_BEND_RLF_RECTANGULAR—The selected bend reliefs are edited to rectangular reliefs with width and depth dimensions. • PRO_BEND_RLF_OBROUND—The bend relief is edited to an obround relief with width and depth dimensions.
PRO_E_BEND_RELIEF_WIDTH	PRO_VALUE_TYPE_DOUBLE	This element is mandatory when the element PRO_E_BEND_

Element ID	Data Type	Description
		RELIEF_TYPE is set to PRO_BEND_RLF_RECTANGULAR or PRO_BEND_RLF_OBROUND. Specifies the value for width in a bend relief.
PRO_E_BEND_RELIEF_DEPTH_TYPE	PRO_VALUE_TYPE_INT	This element is mandatory when the element PRO_E_BEND_RELIEF_TYPE is set to PRO_BEND_RLF_RECTANGULAR or PRO_BEND_RLF_OBROUND. Specifies the type of depth for bend relief. The valid values are: • PRO_BEND_RLF_DEPTH_BLIND—Creates a relief through the geometry as per the specified value. • PRO_BEND_RLF_DEPTH_UP_TO_BEND— Creates a relief up to the bend. • PRO_BEND_RLF_DEPTH_TAN_TO_BEND—This depth type is applicable only for bend relief type PRO_BEND_RLF_OBROUND. Creates a relief tangential to the bend.
PRO_E_BEND_RELIEF_DEPTH	PRO_VALUE_TYPE_DOUBLE	This element is mandatory when the element PRO_E_BEND_RELIEF_DEPTH_TYPE is set to PRO_BEND_RLF_DEPTH_BLIND. Specifies the value for depth in a bend relief. The depth is measured from the edge of the bend.
PRO_E_BEND_RELIEF_LENGTH_TYPE	PRO_VALUE_TYPE_INT	This element is mandatory and it specifies the type of the relief length. It is defined by the enumerated data type ProBendRlfLengthType and the valid values follow: • PRO_BEND_RLF_LENGTH_NOT_USED • PRO_BEND_RLF_LENGTH_BLIND—Creates the bend reliefs with a length of the specified value. • PRO_BEND_RLF_LENGTH_TO_NEXT—Creates the bend

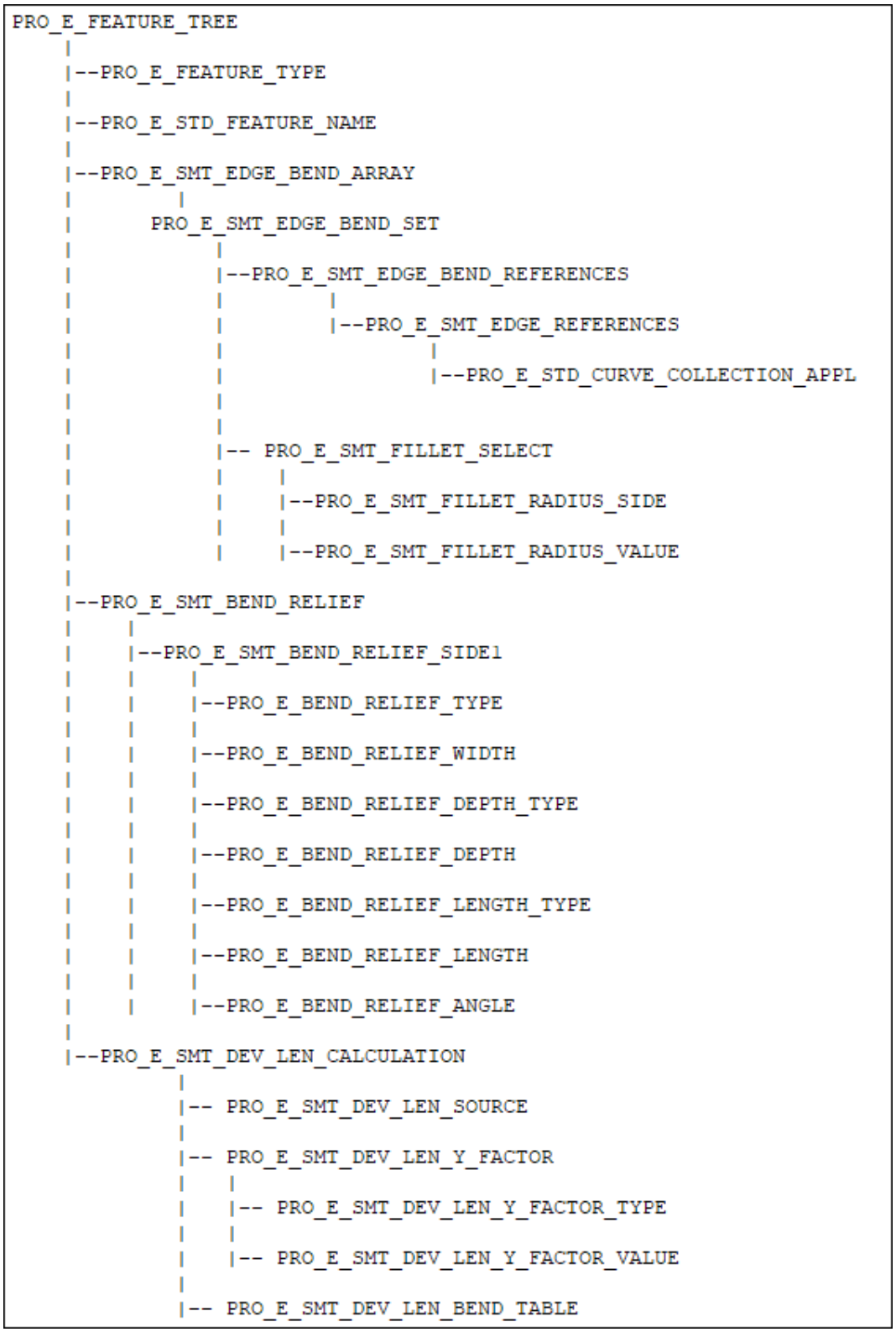
Element ID	Data Type	Description
		reliefs with a length to the next surface. • PRO_BEND_RLF_LENGTH_THROUGH_ALL—Creates the bend reliefs through all surfaces. • PRO_BEND_RLF_LENGTH_TYPE_PARAM—Uses the SMT_DFLT_BEND_REL_LENGTH_TYPE parameter value.
PRO_E_BEND_RELIEF_LENGTH	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the relief length. This element is mandatory when the length type is PRO_BEND_RLF_LENGTH_BLIND. Only zero or positive values are allowed
PRO_E_EDIT_BEND_RELIEF_TABLE	Array	This element along with its child elements is reserved for internal use.

Edge Bend Feature

The Edge Bend feature allows you to round sharp edges of a sheet metal.

The element tree for the Edge Bend feature is documented in the header file ProSmtEdgeBend.h and is shown in the following figure:

Feature Element Tree for Edge Bend Feature



The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature. The value of this feature must be PRO_FEAT_EDGE_BEND.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Edge_Bend_1.
PRO_E_SMT_EDGE_BEND_ARRAY	Array	An array element of single or multiple Edge Bend sets.
PRO_E_SMT_EDGE_BEND_SET	Compound	This compound element contains a single edge bend set.
PRO_E_SMT_EDGE_BEND_REFERENCES	Array	An array element of surface edges, that form an edge bend set.
PRO_E_SMT_EDGE_REFERENCES	Compound	This compound element defines the collection of surface edge or a curve.
PRO_E_STD_CURVE_COLLECTION_APPL	PRO_VALUE_TYPE_POINTER	Specifies the sharp edges to be rounded.
PRO_E_SMT_FILLET_SELECT	Compound	This compound element defines the bend properties and the value of the bend radius.
PRO_E_SMT_FILLET_RADIUS_SIDE	PRO_VALUE_TYPE_INT	Specifies the fillet side using the enumerated data type ProSmdRadType. The valid values for this element are as follows: • PRO_BEND_RAD_OUTSIDE—Applies the bend radius to the outer surface of the bend. • PRO_BEND_RAD_INSIDE—Applies the bend radius to the inner surface of the bend. • PRO_BEND_RAD_PARAMETER—Applies the bend radius at the dimension location set by the SMT_DFLT_RADIUS_SIDE parameter in Creo Parametric.
PRO_E_SMT_FILLET_RADIUS_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the bend radius.

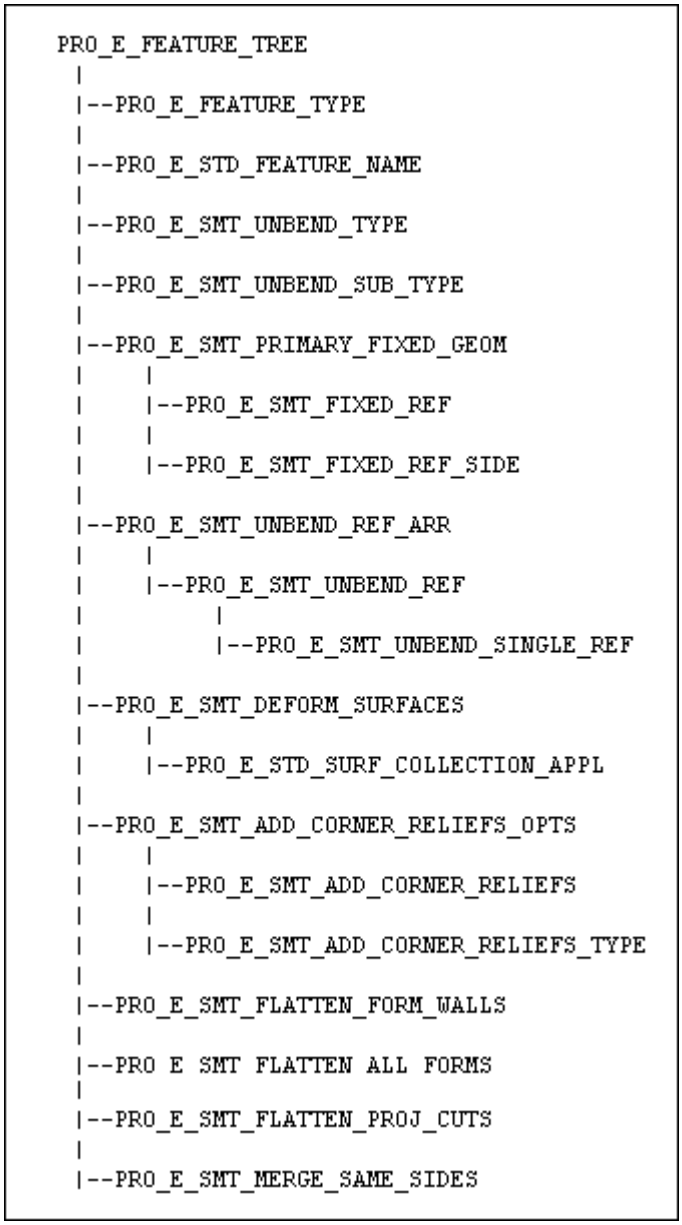
Element ID	Data Type	Description
PRO_E_SMT_BEND_RELIEF	Compound	This compound element defines the bend relief at the edges. Refer to the section Bend Relief Elements on page 1398 for more information.
PRO_E_SMT_DEV_LEN_ CALCULATION	Compound	This compound element defines the method used to calculate the Developed Length dimensions for bends. For more information see the section The Element Subtree for Length Calculation on page 1327.

Unbend Feature

The unbend feature allows you to unbend one or more cylinder based curvature surfaces such as bends or curved walls in a sheet metal part.

The element tree for the Unbend feature is documented in the header file `ProRegularUnbend.h` and is shown in the following figure:

Feature Element Tree for Unbend Feature



The elements in this tree are described as follows:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature. The value of this feature must be PRO_FEAT_UNBEND.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Unbend_1.
PRO_E_SMT_UNBEND_TYPE	PRO_VALUE_TYPE_INT	Creates a regular unbend feature using the enumerated data type ProSmtUnbendType. This

Element ID	Data Type	Description
		element takes the valid value PRO_SMT_REGULAR_UNBEND.
PRO_E_SMT_UNBEND_SUB_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of unbend execution using the enumerated data type ProUnbendSubType which has the following values: • PRO_UNBEND_ALL—Specifies that all the curved surfaces and edges must be automatically selected for unbend. • PRO_UNBEND_SELECTED—Specifies that the curved surfaces and edges must be manually selected for unbend.
PRO_E_SMT_PRIMARY_FIXED_GEOM	Compound	This compound element defines a surface or an edge that remains fixed during the unbending.
PRO_E_SMT_FIXED_REF	PRO_VALUE_TYPE_SELECTION	Specifies a surface or edge that remains fixed during unbending.
PRO_E_SMT_FIXED_REF_SIDE	PRO_VALUE_TYPE_INT	This element is applicable only if a sharp edge is selected as the fixed reference in the element PRO_E_SMT_FIXED_REF. Flips the edge and one of the surfaces the edge lies in between, to remain fixed during unbending, using the enumerated data type ProSmtFixedRefSide. The valid values for this element are: • PRO_SMT_FIXED_SIDE_ONE—First side from the edge will be fixed. • PRO_SMT_FIXED_SIDE_TWO—Second side from the edge will be fixed.
PRO_E_SMT_UNBEND_REF_ARR	Array	An array element defining the list of edges to unbend.
PRO_E_SMT_UNBEND_REF	Compound	This compound element defines collection of geometry to be unbent.
PRO_E_SMT_UNBEND_SINGLE_REF	PRO_VALUE_TYPE_SELECTION	Specifies edge or surface or intent surface or intent chain to unbend.
PRO_E_SMT_DEFORM_SURFACES	Compound	This compound element defines the deformation surfaces.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_POINTER	Collects the surfaces to be used as deformation areas.
PRO_E_SMT_ADD_CORNER_RELIEFS_OPTS	Compound	This compound element defines the relief parameters.

Element ID	Data Type	Description
PRO_E_SMT_ADD_CORNER_RELIEFS	PRO_VALUE_TYPE_BOOLEAN	Specifies if the relief geometry must be created. The valid values for this element are: • PRO_B_TRUE • PRO_B_FALSE
PRO_E_SMT_ADD_CORNER_RELIEFS_TYPE	PRO_VALUE_TYPE_INT	Specifies the corner relief settings using the enumerated data type ProSmtAddCornRelType. The valid values for this element are: • PRO_SMT_ADD_CORN_REL_UNDEF—Specifies that the corner relief parameter is undefined. • PRO_SMT_ADD_CORN_REL_BY_FLAT_PAT—Creates the corner relief geometry on the model. • PRO_SMT_ADD_CORN_REL_BY_PARAMS—Does not creates the corner relief geometry on the model.
PRO_E_SMT_FLATTEN_FORM_WALLS	PRO_VALUE_TYPE_BOOLEAN	Specifies if the walls adjusted to form geometry must be unbent. When forms are also flattened, geometry is first unbent. The valid values are: • PRO_B_TRUE • PRO_B_FALSE **Note** You can specify the value for this element only if the enumerated data type ProUnbendSubTypeis set to the value PRO_UNBEND_ALL.
PRO_E_SMT_FLATTEN_ALL_FORMS	PRO_VALUE_TYPE_BOOLEAN	Specifies if all the forms in the model must be flattened. The valid values are:

Element ID	Data Type	Description
		• PRO_B_TRUE • PRO_B_FALSE **Note** You can specify the value for this element only if the enumerated data type ProSmtUnbendTypeis set to the value PRO_SMT_FLAT_PATTERN.
PRO_E_SMT_FLATTEN_PROJ_CUTS	PRO_VALUE_TYPE_BOOLEAN	Specifies if cuts must be projected to the flattened form.
PRO_E_SMT_MERGE_SAME_SIDES	PRO_VALUE_TYPE_BOOLEAN	Specifies if the side surfaces located in the same location must be kept. The valid values for this element are: • PRO_B_TRUE • PRO_B_FALSE **Note** You can specify the value for this element only if the enumerated data type ProSmtUnbendTypeis set to the value PRO_SMT_FLAT_PATTERN.

Flat Pattern Feature

The Flat Pattern feature creates a flattened version of a sheet metal part.

This feature uses the same element tree as the Unbend feature documented in the header file ProRegularUnbend.h. It shares most of the elements with the Unbend feature tree. The following elements are specific to Flat Pattern feature:

- PRO_E_FEATURE_TYPE—Specifies the type of feature. The value of this feature must be PRO_FEAT_FLAT_PAT.
- PRO_E_SMT_UNBEND_TYPE—Creates a flattened sheet metal part using the enumerated data type ProSmtUnbendType. This element takes the valid value PRO_SMT_FLAT_PATTERN.
- PRO_E_SMT_FLATTEN_ALL_FORMS—Specifies if all the forms in the model must be flattened. The valid values are:
  - PRO_B_TRUE

- PRO_B_FALSE

- PRO_E_SMT_MERGE_SAME_SIDES—Specifies if the side surfaces located in the same location must be kept. The valid values for this element are:
  - PRO_B_TRUE
  - PRO_B_FALSE

Bend Back Feature

The bend back feature allows you to return the unbent walls to their formed bent positions in a sheet metal part.

The element tree for the bend back feature is documented in the header file ProSmtBendBack.h and is shown in the following figure:

Element Tree for Bend Back Feature

```
PRO_E_FEATURE_TREE
  |
  |--PRO_E_FEATURE_TYPE
  |
  |--PRO_E_STD_FEATURE_NAME
  |
  |--PRO_E_SMT_BEND_BACK_SUB_TYPE
  |
  |--PRO_E_SMT_PRIMARY_FIXED_GEOM
  |     |
  |     |--PRO_E_SMT_FIXED_REF
  |     |
  |     |--PRO_E_SMT_FIXED_REF_SIDE
  |
  |--PRO_E_SMT_BBACK_REF_ARR
  |     |
  |     |--PRO_E_SMT_BBACK_REF
  |           |
  |           |--PRO_E_SMT_BBACK_SINGLE_REF
  |
  |
  |--PRO_E_SMT_BBACK_FLAT_CONTOURS_ARR
        |
        |--PRO_E_SMT_GEOM_CONTOUR
              |
              |--PRO_E_SMT_GEOM_CONTOUR_GEOM
              |
              |
              |--PRO_E_SMT_GEOM_CONTOUR_EDGE
```

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of feature. The value of this feature must be PRO_FEAT_BEND_BACK.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature. The default value is Bend_Back_1.
PRO_E_SMT_BEND_BACK_SUB_TYPE	PRO_VALUE_TYPE_INT	Specifies if all or selected geometry must be bent. The enumerated data type ProBendBackSubType contains the valid values for this element and are as follows: • PRO_BEND_BACK_ALL — Specifies that all unbent geometry must be bent back. • PRO_BEND_BACK_SELECTED— Specifies that only the specified geometry must be bent back.
PRO_E_SMT_PRIMARY_FIXED_GEOM	Compound	This compound element defines the fixed geometry during the bend back operation.
PRO_E_SMT_FIXED_REF	PRO_VALUE_TYPE_SELECTION	Specifies a surface or edge that remains fixed during the bend back operation.
PRO_E_SMT_FIXED_REF_SIDE	PRO_VALUE_TYPE_INT	Specifies the flip option for the side of the edge that will remain fixed during the bend back operation. The valid values for this element are defined in the enumerated data type ProSmtFixedRefSideand are as follows: • PRO_SMT_FIXED_SIDE_ONE—First edge side of the selected fixed geometry will be fixed. • PRO_SMT_FIXED_SIDE_TWO—Second edge side normal to the selected fixed geometry will be fixed.
PRO_E_SMT_BBACK_REF_ARR	Array	An array element of edges or surfaces to be bent back.
PRO_E_SMT_BBACK_REF	Compound	This compound element defines collection of edges or surfaces to be bent back.
PRO_E_SMT_BBACK_SINGLE_REF	PRO_VALUE_TYPE_SELECTION	Specifies collection of edges, surfaces, intent surfaces and chains to be bent back.
PRO_E_SMT_BBACK_FLAT_CONTOURS_ARR	Array	An array element that specifies which contours that partially intersect a bend line shall remain

Element ID	Data Type	Description
		flat during the bend back operation.
PRO_E_SMT_GEOM_CONTOUR	Compound	This compound element specifies the collection of contour geometry to remain flat during the bend back operation.
PRO_E_SMT_GEOM_CONTOUR_GEOM	PRO_VALUE_TYPE_SELECTION	Specifies the driven or offset sheet metal surface or surfaces from the element PRO_E_SMT_BBACK_SINGLE_REF that form contours.
PRO_E_SMT_GEOM_CONTOUR_EDGE	PRO_VALUE_TYPE_SELECTION	Specifies edges from the element PRO_E_SMT_GEOM_CONTOUR_GEOM that form contours.

Sketch Form Feature

The sketch form feature helps you to create a punch or a piercing using a sketch. You can also select an existing sketch or define an internal one. This sketch based form tool enables you to specify the punch and the piercing depth.

The element tree for the sketch form feature is documented in the header file ProSmtSketchForm.h and is shown in the following figure:

Element Tree for Sketch Form Feature

```
PRO_E_FEATURE_TREE
    |
    |--PRO_E_FEATURE_TYPE
    |
    |--PRO_E_STD_FEATURE_NAME
    |
    |--PRO_E_STD_SECTION
    |
    |--PRO_E_SKETCH_FORM_TYPE
    |
    |--PRO_E_SKETCH_FORM_DEPTH_PUNCH
    |
    |--PRO_E_SKETCH_FORM_DEPTH_PIERCING
    |
    |--PRO_E_SKETCH_FORM_DIR
    |
    |--PRO_E_EXT_COMP_DRFT_ANG
    |     |
    |     |--PRO_E_EXT_DRFT_ANG
    |     |
    |     |--PRO_E_EXT_DRFT_ANG_VAL
    |
    |-- PRO_E_SRF_END_ATTRIBUTES
    |
    |--PRO_E_FORM_PUNCH_MATERIAL_SIDE
    |
    |--PRO_E_SMT_FILLET_INTERSECT
    |     |
    |     |--PRO_E_SMT_FILLET_RADIUS_USEFLAG
    |     |
    |     |--PRO_E_SMT_FILLET_RADIUS_SIDE
    |     |
    |     |--PRO_E_SMT_FILLET_RADIUS_VALUE
    |
    |--PRO_E_SMT_FILLET_QUILT
    |     |
    |     |--PRO_E_SMT_FILLET_RADIUS_USEFLAG
    |     |
    |     |--PRO_E_SMT_FILLET_RADIUS_SIDE
    |     |
    |     |--PRO_E_SMT_FILLET_RADIUS_VALUE
    |
    |--PRO_E_SMT_TRIM_FORM_SIDES
```

The following table describes the elements in the element tree for the Sketch Form feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_SMT_SKETCH_FORM.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name for the sheet metal sequence. The default value is Sketched_Form_1.
PRO_E_STD_SECTION	Compound	Mandatory element.
PRO_E_SKETCH_FORM_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the sketched form feature. The valid values for this element are defined in the enumerated type ProSketchFormType: • PRO_SMT_SKETCH_FORM_TYPE_PUNCH— Specify the value 1 if you want to select punch as your form type. • PRO_SMT_SKETCH_FORM_TYPE_PIERCING— Specify the value 2 if you want to select piercing as your form type.
PRO_E_SKETCH_FORM_DEPTH_PUNCH	PRO_VALUE_TYPE_DOUBLE	Defines the depth of the penetration of the punching operation. **Note** Specify a value for this element only if the element PRO_E_SKETCH_FORM_TYPE is set to PRO_SMT_SKETCH_FORM_TYPE_PUNCH.
PRO_E_SKETCH_FORM_DEPTH_PIERCING	PRO_VALUE_TYPE_DOUBLE	Defines the depth of the penetration of the piercing operation. **Note** Specify a value for this element only if the element PRO_E_SKETCH_FORM_TYPE is set to PRO_SMT_SKETCH_FORM_TYPE_PIERCING.
PRO_E_SKETCH_FORM_DIR	PRO_VALUE_TYPE_INT	This element changes the direction of the form. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_B_TRUE— Specifies that the direction of the form is changed. • PRO_B_FALSE— Specifies that the direction of the form is not changed.
PRO_E_EXT_COMP_DRFT_ANG	Compound	This compound element defines the parameters for the taper angle. **Note** Use this element only if the element PRO_E_SKETCH_FORM_TYPE is set to PRO_SMT_SKETCH_FORM_TYPE_PUNCH.
PRO_E_EXT_DRFT_ANG	PRO_VALUE_TYPE_INT	Specifies the addition of a taper to the sketch form feature. The valid values for this element are defined in the enumerated type ProExtDrftAng and are as follows: • PRO_EXT_DRFT_ANG_NO_DRAFT— Specifies that the feature has no draft angle or taper. • PRO_EXT_DRFT_ANG_DRAFT— Specifies that the feature has a draft angle or taper. **Note** Use this element only if the element PRO_E_SKETCH_FORM_TYPE is set to PRO_SMT_SKETCH_FORM_TYPE_PUNCH.
PRO_E_EXT_DRFT_ANG_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the tapering of the geometry by the specified value. **Note** Use this element only if the element PRO_E_SKETCH_FORM_TYPE is set to PRO_SMT_SKETCH_FORM_TYPE_PUNCH.
PRO_E_SRF_END_ATTRIBUTES	PRO_VALUE_TYPE_INT	This element caps the sketch plane and offset surface of the form

Element ID	Data Type	Description
		feature. The valid values for this element are defined in the enumerated type `ProExtSurfEndAttr` and are as follows: • `PRO_EXT_SURF_END_ATTR_OPEN`— Specifies that the sketch plane and the offset surface will not be capped. • `PRO_EXT_SURF_END_ATTR_CAPPED`— This is the default value. Specifies that the sketch plane and the offset surface will be capped. **Note** Use this element only if the element `PRO_E_SKETCH_FORM_TYPE` is set to `PRO_SMT_SKETCH_FORM_TYPE_PUNCH`.
`PRO_E_FORM_PUNCH_MATERIAL_SIDE`	`PRO_VALUE_TYPE_INT`	This element flips the material deformation direction for the punching operation. The valid values for this element are defined in the enumerated type `ProSmdPunchMatSide`, and are as follows: • `PRO_SMT_PUNCH_MAT_OUTSIDE`— Specifies that the punching operation takes place on the outer side. • `PRO_SMT_PUNCH_MAT_INSIDE`— Specifies that the punching operation takes place on the inner side. **Note** Use this element only if the element `PRO_E_SKETCH_FORM_TYPE` is set to `PRO_SMT_SKETCH_FORM_TYPE_PUNCH`.
`PRO_E_SMT_FILLET_INTERSECT`	`Compound`	This compound element specifies an option to round the placement sharp edges that lie on the placement references and are created by the intersection of the

Element ID	Data Type	Description
		sheet metal geometry with the quilt. For more information on the elements related to PRO_E_SMT_FILLET_INTERSECT, refer to the section Sub Elements of PRO_E_SMT_FILLET_INTERSECT and PRO_E_SMT_FILLET_QUILT on page 1422.
PRO_E_SMT_FILLET_QUILT	PRO_VALUE_TYPE_INT	This compound element specifies the option to round the non placement sharp edges that do not lie on the placement references and are created by the deformation of the sheet metal geometry based on the quilt. For more information on the elements related to PRO_E_SMT_FILLET_QUILT, refer to the section Sub Elements of PRO_E_SMT_FILLET_INTERSECT and PRO_E_SMT_FILLET_QUILT on page 1422.
PRO_E_SMT_TRIM_FORM_SIDES	PRO_VALUE_TYPE_INT	Trim edges of sheared form. Specifies if Creo Parametric applies trimming of sheetmetal side surfaces during form feature generation. The valid values for this element follow: • PRO_B_TRUE • PRO_B_FALSE

Sub Elements of PRO_E_SMT_FILLET_INTERSECT and PRO_E_SMT_FILLET_QUILT

The following table lists all the elements that are common to the compound elements PRO_E_SMT_FILLET_INTERSECT and PRO_E_SMT_FILLET_QUILT.

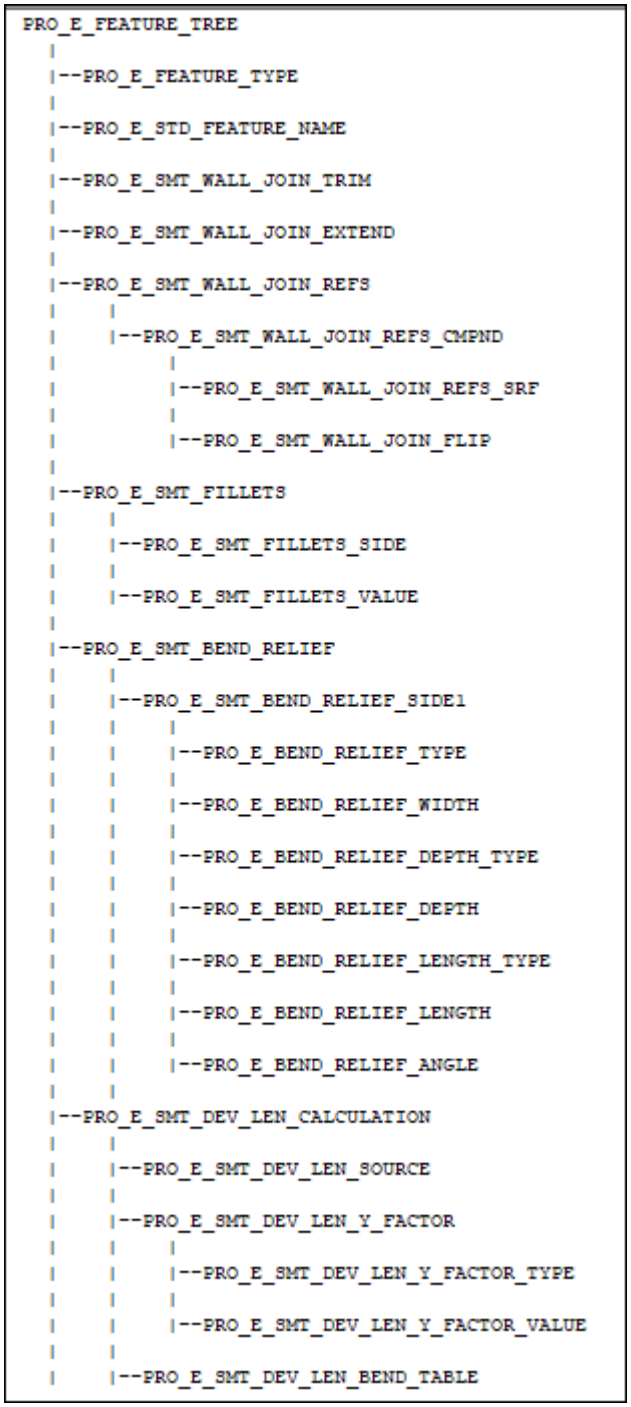
Element ID	Data Type	Description
PRO_E_SMT_FILLET_RADIUS_USEFLAG	PRO_VALUE_TYPE_INT	Specifies whether a fillet radius is used. The valid values for this element are: • PRO_B_TRUE— Specifies that a fillet is used. • PRO_B_FALSE— Specifies that the fillet is not used.
PRO_E_SMT_FILLET_RADIUS_SIDE	PRO_VALUE_TYPE_INT	Specifies the radius direction. The values for this element are specified in the enumerated type ProSmdRadType, are as follows: • PRO_BEND_RAD_OUTSIDE— Specifies that the radius is applied to the outside of the sheet metal geometry. • PRO_BEND_RAD_INSIDE— Specifies that the radius is applied to the inside of the sheet metal geometry. **Note** Use this element only if the element PRO_E_SKETCH_FORM_TYPE is set to PRO_SMT_SKETCH_FORM_TYPE_PUNCH.
PRO_E_SMT_FILLET_RADIUS_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the radius value.

Join Feature

The Join feature helps you to connect two intersecting walls in a sheet metal part. You can trim the non intersecting portions of the walls as well as add a bend and bend relief at the intersection. You can also control the location of the intersection in which the feature would be created.

The element tree for the Join feature is documented in the header file ProSmtJoinWalls.h and is shown in the following figure:

Element Tree for Join Feature



The following table describes the elements in the element tree for the Join feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the sheet metal feature. The valid value for this element is PRO_FEAT_JOIN_WALLS.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name for the sheet metal feature. The default value is JOIN_1.
PRO_E_SMT_WALL_JOIN_TRIM	PRO_VALUE_TYPE_INT	Specifies an option for trimming the non intersecting geometry. The valid values for this element are defined in the enumerated type ProSmtJoinWallsTrimType and are as follows: • PRO_INTWLS_TRIM_OPEN_CUTS— Specifies the original intersecting walls without being trimmed. • PRO_INTWLS_TRIM_BNDR_EXTS— Specifies that the non intersection portions up to the intersection between the surfaces will be removed. • PRO_INTWLS_TRIM_BEND_LINE— Specifies that the non intersecting surfaces will be trimmed up to the bend.
PRO_E_SMT_WALL_JOIN_EXTEND	PRO_VALUE_TYPE_INT	The valid values for this element are defined in the enumerated type ProSmtJoinWallsExtType and are as follows: • PRO_INTWLS_EXT_LINE_TO_INT — Extends the intersection line to the intersection area. • PRO_INTWLS_EXT_LINE_TO_ALL— Extends the intersection line to the intersection plane. **Note** The intersecting walls must be planar.
PRO_E_SMT_WALL_JOIN_REFS	Array	An array element of only two surfaces, that form a join feature set.
PRO_E_SMT_WALL_JOIN_REFS_CMPND	Compound	This compound element defines the collection of geometry to be joined.
PRO_E_SMT_WALL_JOIN_REFS_SRF	PRO_ELEM_TYPE_SELECT	Select the surfaces which are to be connected by the join feature.

Element ID	Data Type	Description
PRO_E_SMT_WALL_JOIN_FLIP	PRO_VALUE_TYPE_INT	This element flips the wall join direction. The selection point of each wall determines which side of the walls will be kept.
PRO_E_SMT_FILLETS	Compound	This compound element defines the bend types of the sheet metal wall and the value of bend radius.
PRO_E_SMT_FILLETS_SIDE	PRO_VALUE_TYPE_INT	Specifies the fillet side. The valid values for this element are defined in the enumerated type ProSmdRadType and are as follows: • PRO_BEND_RAD_OUTSIDE— Applies the bend radius to the outer surface of the bend. • PRO_BEND_RAD_INSIDE— Applies the bend radius to the inner surface of the bend. • PRO_BEND_RAD_PARAMETER— Applies the bend radius at the dimension location set by the SMT_DFLT_RADIUS_SIDE parameter in Creo Parametric.
PRO_E_SMT_FILLETS_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the bend radius.
PRO_E_SMT_BEND_RELIEF	Compound	This compound element defines the bend relief at the edges. For more information see the section Bend Relief Elements on page 1398.
PRO_E_SMT_DEV_LEN_CALCULATION	Compound	This compound element defines the method used to calculate the Developed Length dimensions for bends. For more information see the section The Element Subtree for Length Calculation on page 1327.

Twist Wall Feature

The twist wall feature enables you to create a spiral or coil-shaped section of sheet metal. The twist wall can be attached to a straight edge on an existing planar wall. The twist wall typically serves as a transition between two areas of sheet metal because it can change the plane of a sheet metal part. The twist can be rectangular or trapezoidal.

The element tree for the twist wall feature is documented in the header file ProSmtTwist.h and is shown in the following figure:

Element Tree for Twist Wall Feature

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_SMT_TWIST_ATT_EDGE
   |
   |--PRO_E_SMT_TWIST_TYPE
   |
   |--PRO_E_SMT_TWIST_TRIM_EDGES
   |     |
   |     |--PRO_E_SMT_TWIST_SIDE_1_OFFSET
   |     |     |
   |     |     |--PRO_E_SMT_TWIST_OFFSET_TYPE
   |     |     |
   |     |     |--PRO_E_SMT_TWIST_OFFSET_VAL
   |     |
   |     |--PRO_E_SMT_TWIST_SIDE_2_OFFSET
   |     |     |
   |     |     |--PRO_E_SMT_TWIST_OFFSET_TYPE
   |     |     |
   |     |     |--PRO_E_SMT_TWIST_OFFSET_VAL
   |
   |--PRO_E_SMT_TWIST_AXIS_POINT
   |     |
   |     |--PRO_E_SMT_TWIST_POINT_TYPE
   |     |
   |     |--PRO_E_SMT_TWIST_START_WIDTH_VAL
   |     |
   |     |--PRO_E_SMT_TWIST_ATT_POINT_REF
   |
   |--PRO_E_SMT_TWIST_ANGLE_VAL
   |
   |--PRO_E_SMT_TWIST_WALL_LENGTH_VAL
   |
   |--PRO_E_SMT_TWIST_END_WIDTH
   |     |
   |     |--PRO_E_SMT_TWIST_END_WIDTH_TYPE
   |     |
   |     |--PRO_E_SMT_TWIST_END_WIDTH_VAL
   |
   |--PRO_E_SMT_TWIST_DEV_LEN_VAL
```

The following table describes the elements in the element tree for the twist wall feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the sheet metal feature. The valid value for this element is PRO_FEAT_TWIST.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name for the sheet metal feature. The default value is Twist_1.
PRO_E_SMT_TWIST_ATT_EDGE	PRO_VALUE_TYPE_SELECTION	Specifies an edge to which the twist wall will be attached.
PRO_E_SMT_TWIST_TYPE	PRO_VALUE_TYPE_INT	Specifies the width options for the walls. The valid values for this element are defined by the enumerated data type ProSmtTwistType and are as follows: • PRO_SMT_TWIST_TRIM_EDGES—Calculates the wall width from the ends of the attachment edges. The walls are offset by the specified value from the attachment ends. • PRO_SMT_TWIST_TYPE_PNT—Calculates and centers the wall width from the twist axis by the specified dimension.
PRO_E_SMT_TWIST_TRIM_EDGES	Compound	Specifies a compound element which defines options for twist wall ends. This element is available when the value of the element PRO_E_SMT_TWIST_TYPE is set to PRO_SMT_TWIST_TRIM_EDGES.
PRO_E_SMT_TWIST_SIDE_1_OFFSET PRO_E_SMT_TWIST_SIDE_2_OFFSET	Compound	Specifies the trim option and offset value for the first and second direction of wall ends. The elements PRO_E_SMT_TWIST_OFFSET_TYPE and PRO_E_SMT_TWIST_OFFSET_VAL are common to the compound elements in both directions.
PRO_E_SMT_TWIST_OFFSET_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of trim for the first and second direction using the enumerated data type

Element ID	Data Type	Description
		ProSmtTwistOffsetType. The valid values are: • PRO_TWIST_OFFSET_TYPE_TO_END—Specifies that the ends of the twist wall are set at the end edges of the attachment points. • PRO_TWIST_OFFSET_TYPE_BLIND—Specifies that the wall ends should be trimmed or extended from the end edges of the attachment points in specified direction.
PRO_E_SMT_TWIST_OFFSET_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the offset value. This element is applicable when the value of the element PRO_E_SMT_TWIST_OFFSET_TYPE is set to PRO_TWIST_OFFSET_TYPE_BLIND.
PRO_E_SMT_TWIST_AXIS_POINT	Compound	Specifies a compound element which defines options for twist axis. This element is available when the value of the element PRO_E_SMT_TWIST_TYPE is set to PRO_SMT_TWIST_TYPE_PNT.
PRO_E_SMT_TWIST_POINT_TYPE	PRO_VALUE_TYPE_INT	Specifies the location of the twist axis using the enumerated data type ProSmtTwistPointType. The valid values are: • PRO_SMT_TWIST_MID_PNT—Specifies that the twist axis is located at the center of the wall width. • PRO_SMT_TWIST_DTM_PNT—Specifies that the twist axis is located on the specified datum point.
PRO_E_SMT_TWIST_START_WIDTH_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the width of the start wall.
PRO_E_SMT_TWIST_ATT_POINT_REF	PRO_VALUE_TYPE_SELECTION	This element is available when the value of the element PRO_E_SMT_TWIST_POINT_TYPE is set to PRO_SMT_TWIST_DTM_PNT.

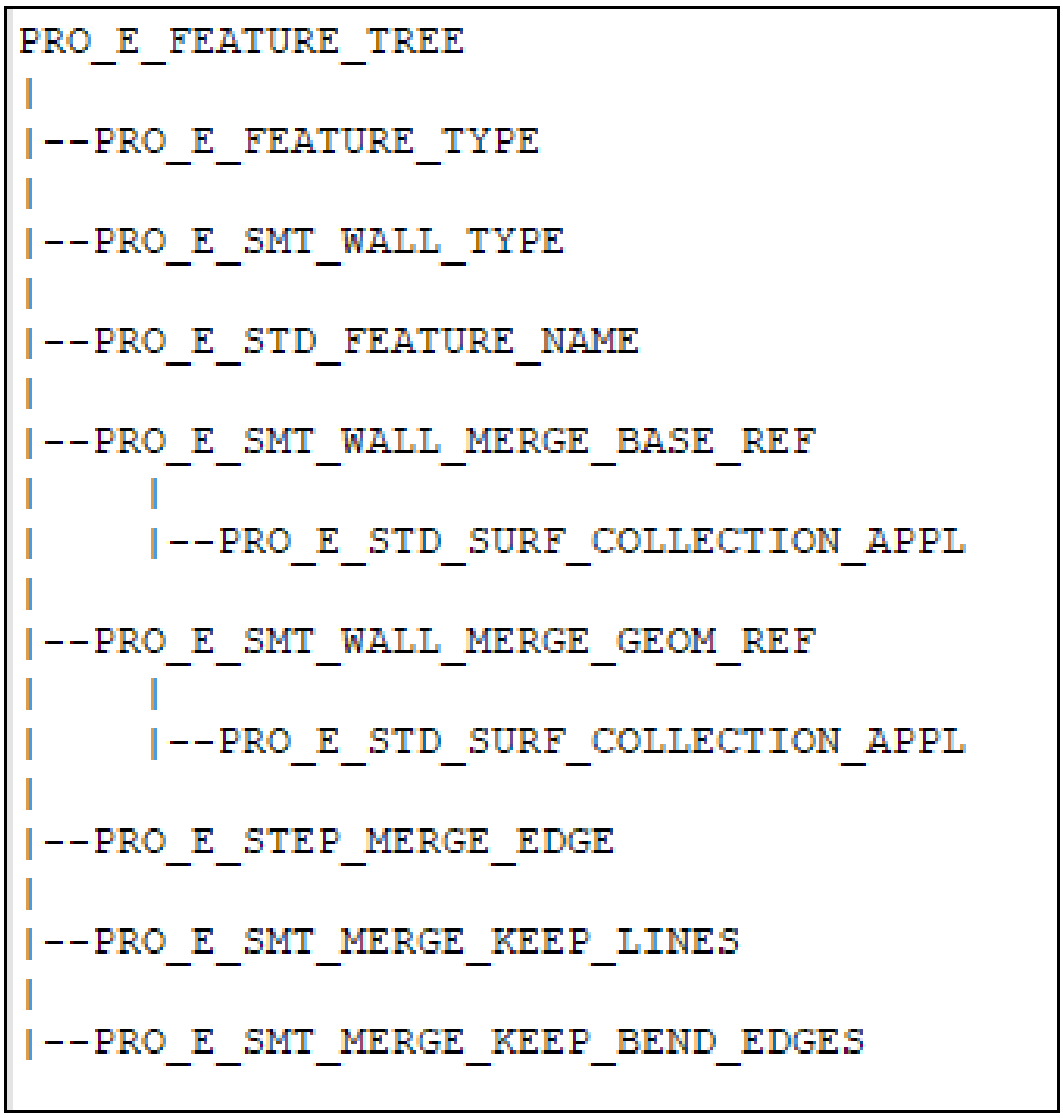
Element ID	Data Type	Description
		Specifies a datum point on the attachment edge. The centerline of the twist wall passes through this datum point. The centerline of the twist axis is perpendicular to the start edge and coplanar with the existing wall.
PRO_E_SMT_TWIST_ANGLE_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the rotation angle of the twist wall.
PRO_E_SMT_TWIST_WALL_LENGTH_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the length of the twist wall, measured from the attachment edge to the end of the twist axis.
PRO_E_SMT_TWIST_END_WIDTH	Compound	Specifies a compound which defines options to change the width of the end wall.
PRO_E_SMT_TWIST_END_WIDTH_TYPE	PRO_VALUE_TYPE_INT	Specifies the width option for the end wall using the enumerated data type ProSmtTwistEndWidthType. The valid values are: • PRO_TWIST_END_WIDTH_SAME_AS_START—Specifies that the width of the end wall must be same as the start wall. • PRO_TWIST_END_WIDTH_BLIND—Specifies that the width of the end wall must be set to the specified value.
PRO_E_SMT_TWIST_END_WIDTH_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the width of the end wall.
PRO_E_SMT_TWIST_DEV_LEN_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the length of wall in unbent state.

Merge Wall Feature

The merge wall feature enables you to collect unattached walls to merge them together into one piece using the base reference collector. The edges between certain pieces of walls can be excluded from the merge operation to keep the corresponding areas disconnected.

The element tree for the merge wall feature is documented in the header file ProSmtMergeWalls.h and is shown in the following figure:

Element Tree for Merge Wall Feature



The following table describes the elements in the element tree for the merge wall feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the sheet metal feature. The valid value for this element is PRO_FEAT_WALL.
PRO_E_SMT_WALL_TYPE	PRO_VALUE_TYPE_INT	Specifies the wall type for the sheet metal feature. The default value is specified by the enumerated typeProSmtWallWallType and the valid value is PRO_SMT_WALL_TYPE_MERGE.

Element ID	Data Type	Description
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the feature name of the sheet metal feature.
PRO_E_SMT_WALL_MERGE_BASE_REF	Compound	Specifies a collection of surfaces for merging with the base wall.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Specifies the selection of merge geometry that can be selected to be designated as sheet metal design objects.
PRO_E_SMT_WALL_MERGE_GEOM_REF	Compound	Specifies the surfaces of one or more unattached flat walls to merge with the base wall.
PRO_E_STEP_MERGE_EDGE	PRO_VALUE_TYPE_SELECTION	Specifies the excluded edges that are included by the merge of the surfaces.
PRO_E_SMT_MERGE_KEEP_LINES	PRO_VALUE_TYPE_BOOLEAN	Controls the visibility of merged edges on surface joints. The valid values for this element are: • Pro_B_True—Merged edges are visible on surface joints. • Pro_B_False—Merged edges are not visible on surface joints.
PRO_E_SMT_MERGE_KEEP_BEND_EDGES	PRO_VALUE_TYPE_BOOLEAN	Controls the ability to keep edges between the bend surfaces while merging surfaces in the merged walls. The valid values are as follows: • Pro_B_True—Keep edges of bend surfaces between existing bend surfaces in the merged walls • Pro_B_False—Does not keep edges of bend surfaces in the merged walls.

Recognizing Sheet Metal Design Objects

From Creo Parametric 4.0 F000 onward, the Sheetmetal Design items are created as design objects. Bends, bend reliefs, corner reliefs, corner seams, and forms are sheet metal design objects. The Recognition commands enable you to tag surfaces as sheet metal design objects or not sheet metal design objects. The following sheet metal objects can be recognized as, or recognized as not, a sheet metal design object:

- Bends
- Bend reliefs
- Corner reliefs

- Corner seams
- Forms

Note

In the `ProSmtRecognition.h` element tree, you can tag objects of the same type at a time as sheet metal design object or not sheet metal design object. This means only one compound element for objects of the same type can be defined at a time for a feature.

The element tree to recognize as sheet metal design objects or not sheet metal design objects is documented in the header file `ProSmtRecognition.h` and is shown in the following figure:

Element Tree for Recognizing Sheetmetal Features

```
PRO_E_FEATURE_TREE
|
|--PRO_E_FEATURE_TYPE
|
|--PRO_E_STD_FEATURE_NAME
|
|--PRO_E_SMT_BND_RLF_RCG
|     |
|     |--PRO_E_SMT_RECOGNITION_SEL_TYPE
|     |
|     |--PRO_E_SMT_PIO_SCOPE
|
|--PRO_E_SMT_CRN_RLF_RCG
|     |
|     |--PRO_E_SMT_RECOGNITION_SEL_TYPE
|     |
|     |--PRO_E_SMT_PIO_SCOPE
|
|--PRO_E_SMT_BEND_RCG
|     |
|     |--PRO_E_SMT_RECOGNITION_SEL_TYPE
|     |
|     |--PRO_E_SMT_PIO_SCOPE
|
|--PRO_E_SMT_CRN_SEAM_RCG
|     |
|     |--PRO_E_SMT_RECOGNITION_SEL_TYPE
|     |
|     |--PRO_E_SMT_PIO_SCOPE
|
|--PRO_E_SMT_FORM_RCG
|     |
|     |--PRO_E_SMT_RECOGNITION_SEL_TYPE
|     |
|     |--PRO_E_SMT_RCG_FORM_AS_ONE
|     |
|     |--PRO_E_STD_SURF_COLLECTION_APPL
|     |
|     |---PRO_E_SMT_FORM_BOUND_REFS
|
|--PRO_E_SMT_BND_RLF_UNRCG
|
|--PRO_E_SMT_CRN_RLF_UNRCG
|
|--PRO_E_SMT_BEND_UNRCG
|
|--PRO_E_SMT_CRN_SEAM_UNRCG
|
|--PRO_E_SMT_FORM_UNRCG
```

The following table describes the elements in the element tree:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the sheet metal feature. The valid value for this element is PRO_FEAT_SMT_RECOGNITION.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name for the sheet metal feature.
PRO_E_SMT_BND_RLF_RCG	Compound	Specifies a compound element which defines bend reliefs as sheet metal design objects.
PRO_E_SMT_RECOGNITION_SEL_TYPE	PRO_VALUE_TYPE_INT	Specifies the mode for selecting bend reliefs. See the section Values for PRO_E_SMT_RECOGNITION_SEL_TYPE on page 1438, for more information on valid values.
PRO_E_SMT_PIO_SCOPE	PRO_VALUE_TYPE_SELECTION	Specifies the geometry that can be selected to be designated as sheet metal design objects. You can select driven or offset sheet metal surface, intent surface that contains at least one side of bend relief, a design object which is not bend relief, thickness edge if the bend relief does not contain any surface, bend relief vertex if the bend relief does not contain any surface or edge.
PRO_E_SMT_CRN_RLF_RCG	Compound	Specifies a compound element which defines corner reliefs as sheet metal design objects.
PRO_E_SMT_RECOGNITION_SEL_TYPE	PRO_VALUE_TYPE_INT	Specifies the mode for selecting corner relief. See the section Values for PRO_E_SMT_RECOGNITION_SEL_TYPE on page 1438, for more information on valid values.
PRO_E_SMT_PIO_SCOPE	PRO_VALUE_TYPE_SELECTION	Specifies the corner relief geometry that can be selected to be designated as sheet metal design objects.
PRO_E_SMT_BEND_RCG	Compound	Specifies a compound element which defines bends as sheet metal design objects.
PRO_E_SMT_RECOGNITION_SEL_TYPE	PRO_VALUE_TYPE_INT	Specifies the mode for selecting bends. See the section Values for PRO_E_SMT_RECOGNITION_SEL_TYPE on page 1438, for more information on valid values.
PRO_E_SMT_PIO_SCOPE	PRO_VALUE_TYPE_SELECTION	Specifies the bend geometry that can be selected to be designated as sheet metal design objects. You can select cylindrical bend surface, intent surface that contains a

Element ID	Data Type	Description
		cylindrical bend or a design object which is not a bend.
PRO_E_SMT_CRN_SEAM_RCG	Compound	Specifies a compound element which defines corner seams as sheet metal design objects.
PRO_E_SMT_RECOGNITION_SEL_TYPE	PRO_VALUE_TYPE_INT	Specifies the mode for selecting corner seams. See the section Values for PRO_E_SMT_RECOGNITION_SEL_TYPE on page 1438, for more information on valid values.
PRO_E_SMT_PIO_SCOPE	PRO_VALUE_TYPE_SELECTION	Specifies the corner seam geometry that can be selected to be designated as sheet metal design objects. You can select one or more references from the following: • Side surface that can be associated with a corner seam. • Not Corner Seam design objects. • Planar face or offset surface, which will be considered as the reference for all neighbor corner seams. • Bend surface, which will be considered as the reference for all neighbor corner seams.
PRO_E_SMT_FORM_RCG	Compound	Specifies a compound element which defines forms as sheet metal design objects.
PRO_E_SMT_RECOGNITION_SEL_TYPE	PRO_VALUE_TYPE_INT	Specifies the mode for selecting forms. See the section Values for PRO_E_SMT_RECOGNITION_SEL_TYPE on page 1438, for more information on valid values.
PRO_E_SMT_RCG_FORM_AS_ONE	PRO_VALUE_TYPE_BOOLEAN	Specifies if the selected form geometry must be considered as one form design object.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Specifies the selection of form geometry that can be selected to be designated as sheet metal design objects.
PRO_E_SMT_FORM_BOUND_REFS	PRO_VALUE_TYPE_SELECTION	Specifies the forms on a reference surface. You can select driven or offset sheet metal surfaces or intent surfaces that contain driven or offset sheet metal surfaces.

Element ID	Data Type	Description
PRO_E_SMT_BND_RLF_UNRCG	Compound	Specifies a compound element which defines bend reliefs as not sheet metal design objects. The child elements and their values are same as the PRO_E_SMT_BND_RLF_RCG element.
PRO_E_SMT_CRN_RLF_UNRCG	Compound	Specifies a compound element which defines corner reliefs as not sheet metal design objects. The child elements and their values are same as the PRO_E_SMT_CRN_RLF_RCG element.
PRO_E_SMT_BEND_UNRCG	Compound	Specifies a compound element which defines bends as not sheet metal design objects. The child elements and their values are same as the PRO_E_SMT_BEND_RCG element.
PRO_E_SMT_CRN_SEAM_UNRCG	Compound	Specifies a compound element which defines corner seams as not sheet metal design objects. The child elements and their values are same as the PRO_E_SMT_CRN_SEAM_RCG element.

Element ID	Data Type	Description
PRO_E_SMT_PIO_SCOPE	PRO_VALUE_TYPE_SELECTION	Specifies the corner seam geometry that can be selected to be designated as not sheet metal design objects. You can select one or more references from the following: • Side surface that can be associated with a corner seam. • Corner Seam design objects. • Planar face or offset surface, which will be considered as the reference for all neighbor corner seams. • Bend surface, which will be considered as the reference for all neighbor corner seams.
PRO_E_SMT_FORM_UNRCG	Compound	Specifies a compound element which defines forms as not sheet metal design objects. The child elements and their values are same as the PRO_E_SMT_FORM_RCG element.

Values for PRO_E_SMT_RECOGNITION_SEL_TYPE

This element specifies the mode for selecting bends, bend reliefs, corner reliefs, corner seams, and forms as sheet metal design objects. The valid values are:

- PRO_SMT_RECOGNITION_MANUAL_SEL—Specifies manual selection of the sheetmetal design object.
- PRO_SMT_RECOGNITION_AUTO_SEL—Specifies automatic selection of the sheetmetal design object.

61

Production Applications: Manufacturing

This chapter describes the Creo Parametric TOOLKIT functions for manufacturing operations. Familiarity with Creo NC functions simplifies the use of these manufacturing functions.

Manufacturing Models

Functions Introduced:

- **ProMfgAssemGet()**
- **ProMfgTypeGet()**

You can use the function `ProSolidFeatVisit()` to visit all the components in a manufacturing model. However, this function requires a `ProSolid` (or one of its instances, `ProPart` or `ProAssembly`) handle to the model as input. The function `ProMfgAssemGet()` outputs a `ProAssembly` handle to the top-level assembly in the manufacturing model, given its `ProMfg` handle. This assembly handle can then be passed to `ProSolidFeatVisit()`.

Manufacturing models, like other models in Creo Parametric, are uniquely identified by name and type. However, there are several different varieties of manufacturing models. For example, assembly machining, sheetmetal manufacturing, and mold manufacturing are all types of manufacturing models. The `ProMfg` object is a general purpose, opaque handle used to represent any of the different manufacturing model varieties. To distinguish between the different types of manufacturing model, there are manufacturing subtypes. The complete list of subtypes is as follows:

- `PRO_MFGTYPE_MACH_ASSEM`
- `PRO_MFGTYPE_SHEET_METAL`
- `PRO_MFGTYPE_MOLD`
- `PRO_MFGTYPE_CAST`
- `PRO_MFGTYPE_CMM`

The function `ProMfgTypeGet()` outputs the subtype, given the `ProMfg` handle to the manufacturing object.

Creating a Manufacturing Model

Function Introduced

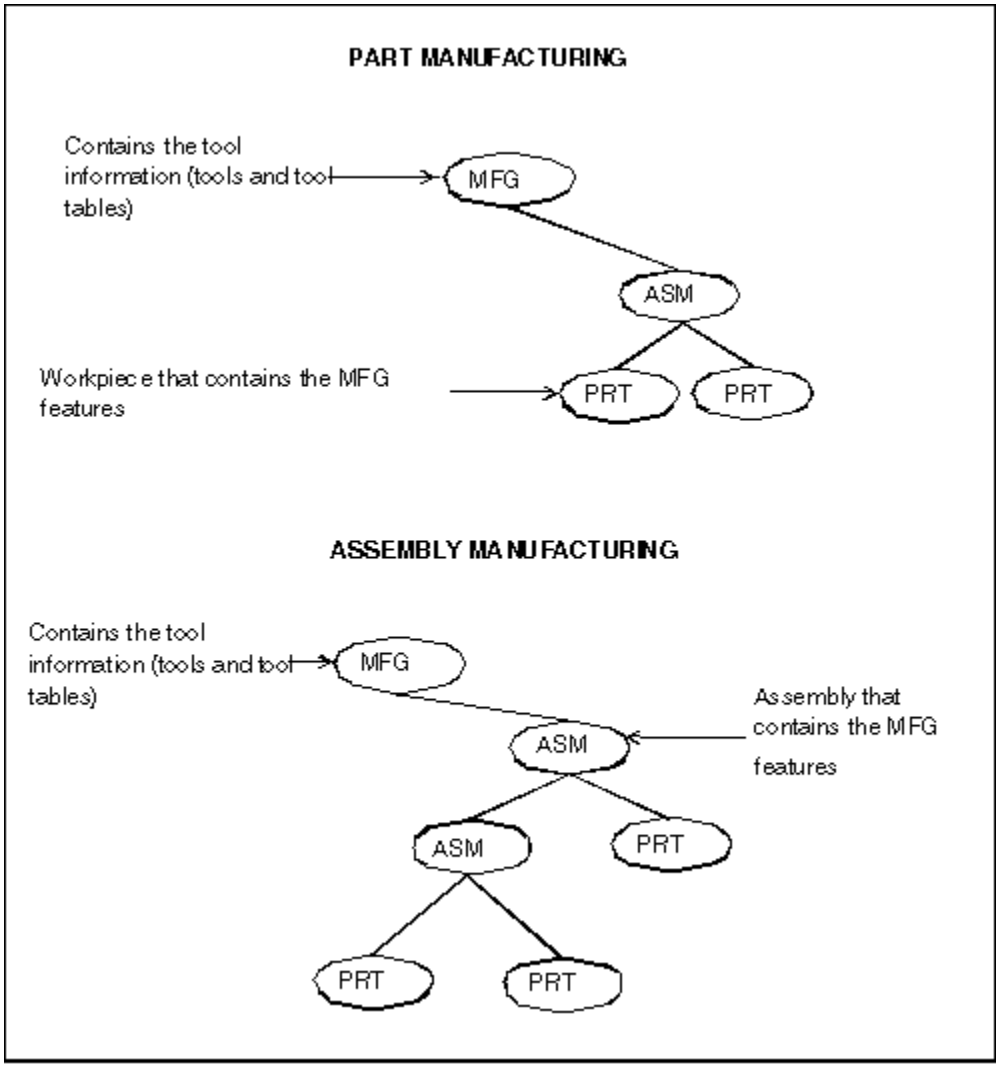
- **ProMfgMdlCreate()**

The function `ProMfgMdlCreate()` outputs an initialized `ProMfg` object handle, given the model name, manufacturing subtype, and name of the reference model. For sheetmetal manufacturing, this should be the sheetmetal workpiece. For all other subtypes, the reference model argument is ignored.

Analyzing a Manufacturing Model

Creo NC has two modes of operation—part and assembly manufacturing. In both cases, the top-level model is an assembly that contains the description of the tools. The following diagram shows the hierarchy of part and assembly manufacturing models.

Part and Assembly Manufacturing Model Hierarchy



In part manufacturing, the storage solid is a part that represents the workpiece or stock, and the design piece is another component at the same level. In assembly manufacturing, the storage solid is the actual assembly representing the design model. The workpiece can be at any level inside this assembly. In both types of manufacturing, the manufacturing operations are described as features of the storage solid.

The important tasks for Creo Parametric TOOLKIT are to traverse the manufacturing assembly components, identify the storage solid that contains the manufacturing operations as its features, and list the manufacturing tools.

This section contains the following subsections:

Traversing Manufacturing Components

Function Introduced:

- **ProAsmcompTypeGet()**

You can visit the components in a manufacturing assembly using the same functions that enable you to visit the components of a regular assembly. For a full description of these functions, see the Assembly: Basic Assembly Access on page 1130 chapter.

The components within a manufacturing assembly perform a variety of different roles. The function `ProAsmcompTypeGet()` provides the role of any model under a manufacturing assembly. The possible roles are as follows:

- `PRO_ASM_COMP_TYPE_NONE`—A regular component (no special manufacturing role)
- `PRO_ASM_COMP_TYPE_WORKPIECE`—A workpiece
- `PRO_ASM_COMP_TYPE_REF_MODEL`—A reference model
- `PRO_ASM_COMP_TYPE_FIXTURE`—A fixture
- `PRO_ASM_COMP_TYPE_MOLD_BASE`—A mold base
- `PRO_ASM_COMP_TYPE_MOLD_COMP`—A mold component
- `PRO_ASM_COMP_TYPE_MOLD_ASSEM`—A mold assembly
- `PRO_ASM_COMP_TYPE_GEN_ASSEM`—A general assembly
- `PRO_ASM_COMP_TYPE_CAST_ASSEM`—A cast assembly
- `PRO_ASM_COMP_TYPE_DIE_BLOCK`—A die block
- `PRO_ASM_COMP_TYPE_DIE_COMP`—A die component
- `PRO_ASM_COMP_TYPE_SAND_CORE`—A sand core
- `PRO_ASM_COMP_TYPE_CAST_RESULT`—A cast result

Identifying the Storage Solid

Functions Introduced:

- **ProMfgSolidGet()**
- **ProMfgFeatureOwnerGet()**

Another important task in using Creo Parametric TOOLKIT for accessing Creo NC models is to find the storage solid inside the manufacturing model. This model is important because the manufacturing operations—workcells, NC sequences, and so on—are represented as features within it.

Manufacturing features are treated like all other features in Creo Parametric TOOLKIT. This enables you to search for NC sequences as you would for solid features. For example, to visit all the workcells of a manufacturing solid, you can use the `ProSolidFeatVisit()` function and filter out any model items not of type `PRO_E_WCELL`.

The function `ProMfgSolidGet()` returns the handle to the storage solid, and `ProMfgFeatureOwnerGet()` returns the component path to the solid from the top-level assembly.

Example 1: Identifying Workcell Features of a NC Model

The sample code in `UgMfgWcellIdentify.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to identify workcell features.

Visiting Manufacturing Tools

Functions Introduced:

- **ProMfgToolVisit()**
- **ProToolTypeGet()**
- **ProToolModelMdlnameGet()**
- **ProToolParamGet()**
- **ProToolAttributesGet()**
- **ProToolVerify()**

Tools are not stored as features of the manufacturing solid, nor as components in the assembly; they are special objects within the top-level manufacturing model, so they need their own visit function. This function is `ProMfgToolVisit()`, which has the same form as other visit functions in Creo Parametric TOOLKIT, except it does not offer the option of a user-defined filter function. You call the action function with an input argument of type `ProTool` to identify the tool.

The other functions analyze the tool being visited.

The function `ProToolTypeGet()` returns the tool type. Examples of the types are as follows:

- PRO_TOOL_DRILL
- PRO_TOOL_TURN
- PRO_TOOL_SAW
- PRO_TOOL_MILL

The ProToolModelMdlnameGet() function outputs the model name and type of a tool, given its ProTool handle.

The function ProToolParamGet() retrieves the value of a specified tool parameter. This yields a value in terms of the type ProParamvalue (see the Core: Parameters on page 210 chapter for more information).

The function ProToolAttributesGet() provides the current setting of several Boolean attributes of the tool in the form of an integer bitmap. Currently, the attributes define whether a solid tool is to be by reference or by copy. See the section Creating NC Sequences on page 1455 for more information on creating tools.

The function ProToolVerify() returns a Boolean showing whether a specified tool handle corresponds to an existing tool.

Creating Manufacturing Objects

This section explains how to create all types of manufacturing feature. This section assumes you have an understanding of element trees for feature creation and Creo NC. For an introduction to element trees, see Element Trees: Principles of Feature Creation on page 764.

Note

You must have a Creo NC license to create manufacturing features using Creo Parametric TOOLKIT.

An important principle of creating manufacturing features is that all the elements required to be defined interactively must also be defined when you create that feature using Creo Parametric TOOLKIT.

As with creating solid features, manufacturing features use element trees to define the feature before you call ProFeatureCreate() to create the feature. However, the method of creating tools is slightly different, as described in the following sections.

Creating Tools

Functions Introduced:

- **ProToolinputAlloc()**
- **ProToolinputTypeSet()**
- **ProToolElemParamAdd()**
- **ProToolElemModelSet()**
- **ProToolinputElemAdd()**
- **ProToolInit()**
- **ProToolCreate()**
- **ProToolinputFree()**
- **ProToolFileRead()**
- **ProToolFileWrite()**

In Creo Parametric, and therefore in Creo Parametric TOOLKIT, tools are not features, and must be created in a slightly different manner for solid and manufacturing features.

Tool creation involves initializing an input structure using a call to `ProToolinputAlloc()`.

You set the tool type (for example, center drill or ream) directly in the input structure using the function `ProToolinputTypeSet()`.

You can then add tool elements to this input structure using a three-step process. First, initialize each element using the function `ProElementAlloc()`. Next, add data to this element using an element-specific function. Finally, add the element to the tool input structure using `ProToolinputElemAdd()`.

As in Creo Parametric, tools can be defined by parameter or by model. To add a parameter to a tool, first allocate the space for a parameter element using a call such as this:

```
status = ProElementAlloc (PRO_E_PARAM, &element);
```

Next, add the parameter to the element using the function `ProToolElemParamAdd()`, then add the element itself to the input structure using the function `ProToolinputElemAdd()`.

The following table lists the parameters required to be defined for each turning tool.

Parameter	Turn	Turn Groove
NOSE_RADIUS	•	•
TOOL_WIDTH	•	•
SIDE_WIDTH	•	
LENGTH	•	•
SIDE_ANGLE	•	•
END_ANGLE	•	•
GAUGE_X_LENGTH	•	•
GAUGE_Z_LENGTH	•	•

Parameter	Turn	Turn Groove
TOOL_MATERIAL	•	•
HOLDER_TYPE	•	

The following table lists the parameters required to be defined for milling tools.

Parameter	Mill	Side Mill	Thread Mill	Groove
CUTTER_DIAM	•	•	•	•
CORNER_RADIUS	•	•		•
CUTTER_WIDTH		•		
SHANK_DIAM		•		
LENGTH	•	•	•	•
INSERT_LENGTH			•	
END_OFFSET			•	
SIDE_ANGLE	•	•		
GAUGE_X_ LENGTH	•	•		
GAUGE_Z_ LENGTH	•	•		•
NUM_OF_TEETH	•	•	•	
TOOL_MATERIAL	•	•		

The following table lists the parameters required to be defined for auxiliary and contouring tools.

Parameter	Auxiliary	Contouring
CUTTER_DIAM	•	•
LENGTH	•	•

The following table lists the parameters required to be defined for holemaking tools.

Parameter	Drill	Csink	Tap	Ream	Center Drill	Bore	Back-Spot
CUTTER_DIAM	•	•	•	•	•	•	•
POINT_ DIAMETER		•	•				
DRILL_ DIAMETER					•		
BODY_ DIAMETER							•
LENGTH	•	•	•	•	•	•	•
CHAMFER_ LENGTH			•				
DRILL_LENGTH					•		
INSERT_ LENGTH							•
TIP_OFFSET		•					
GAUGE_OFFSET		•					
CUTTING_ OFFSET							•
SIDE_ANGLE				•			

Parameter	Drill	Csink	Tap	Ream	Center Drill	Bore	Back-Spot
POINT_ANGLE	•	•			•		
CSINK_ANGLE					•		•
GAUGE_X_LENGTH	•	•	•	•	•	•	•
GAUGE_Z_LENGTH	•	•	•	•	•	•	•
TOOL_MATERIAL	•	•	•	•	•	•	•

Refer to the Creo Parametric NC Manufacturing Help for more information.

Creating a tool using a tool model is similar to the previous process. First, allocate space for an element of type `PRO_E_TOOL_MODEL`. Set the model in the element using the function `ProToolElemModelSet()`, then add it to the input structure using `ProToolinputElemAdd()`. As in Creo NC, you must specify the required number of dimensions within the tool model.

Creating the tool requires two steps. First, initialize a tool handle using `ProToolInit()`. This creates a tool identifier that uniquely defines the tool and is used to reference that tool within the manufacturing model. You pass this identifier, together with the completed input structure, to the function `ProToolCreate()`, which actually creates the tool.

Once the tool has been created, release the memory used by the tool input structure and its associated elements using the function `ProToolinputFree()`.

The following table shows the elements required for tool creation. In this table, the Value column specifies whether the element is required (R) or optional (O).

Element	Description	Value
Name	The name used to identify the tool	R
Type	Mill, drill, and so on	R
Parameters (for parameter-driven tools only)	Tool parameters	R
Model (for solid tools only)	Model that represents the tool	R

The function `ProToolFileRead()` creates a new tool or redefines an existing tool. The input arguments for this function are as follows:

- `tool_handle`—Specify the handle to the tool to be created or redefined.
- `input_file`—Specify the full path and name of the input file that contains all the parameter information about the tool to be created or redefined.

The function `ProToolFileWrite()` writes all information about the tool into a file. Pass the following as input arguments to this function:

- `tool_handle`—Specify the handle to the tool whose information is to be saved.
- `output_file`—Specify the full path and name of the output file where the tool information is to be saved.

Example 2: Creating a Tool from a Solid Model

The sample code in `UgMfgSldToolCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to create a tool from solid model.

Example 3: Creating a Parameter-Driven Tool

The sample code in `UgMfgParamToolCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to create a drilling tool from parameters.

Example 4: Creating a Milling Workcell

The sample code in `UgMfgWcellCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows the process to create a milling workcell. Copy the `mill_d20.xml` file from `<creo_toolkit_loadpoint>/protk_appls/models/mfg` to your working directory to create a workcell using the sample code.

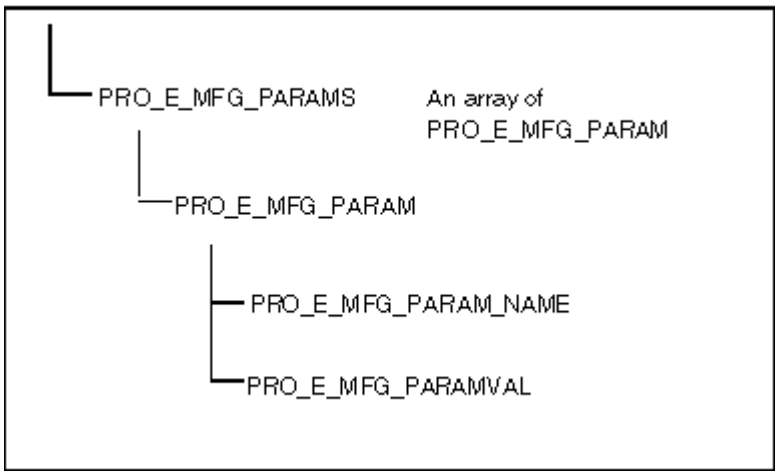
Manufacturing Parameters

In general, there are a number of parameters that are mandatory for a given tool type or NC sequence, and others that are optional. For example, a milling tool requires that its length and diameter be specified, while other parameters such as the number of teeth, or tool material are optional.

The addition of manufacturing parameters to both workcells and operations is optional.

The following figure shows a parameter element subtree.

Parameter Element Subtree



The process of creating a parameter element subtree is the same for workcells, operations, and NC sequences. First, allocate the space for the `PRO_E_MFG_PARAMS` array element. The simplest method of creating the tree is to delay adding the `PRO_E_MFG_PARAMS` element to its parent until you have fully defined the tree. As you define each `PRO_E_MFG_PARAM` element, add it to the `PRO_E_MFG_PARAMS` array using `ProElemtreeElementAdd()`. Use NULL for the element path as each `PRO_E_MFG_PARAM` element is added to the parameter element tree.

The `PRO_E_MFG_PARAM` element itself is a compound element and requires two children to be defined. One is the `PRO_E_MFG_PARAM_NAME` element, a string (not a wide string) that represents the parameter name to define. The other is a `PRO_E_MFG_PARAMVAL` element, which represents the value of the parameter. Depending on the context, this might be an integer, double, or wide string. For example, `CUT_FEED` is represented by a double, whereas `NUMBER_OF_ARC_PNTS` is an integer. Refer to the Creo Parametric NC Manufacturing Help for more information on manufacturing parameters.

Example 4: Creating a Parameter Tree

The sample code in `UgMfgParamTreeCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to create a parameter element tree.

Using External Functions to Define Parameters in the Manufacturing Step Table

Creo Parametric TOOLKIT provides the capability to define customized functions that are available from within the Relations user interface and can be used to define relations for a given item. This capability is described in the section Adding

a Customized Function to the Relations Dialog in Creo Parametric on page 208 in the chapter Core: Relations on page 204. External relation functions can also be used within relations stored in steps in the Manufacturing Step Table.

External functions can be used to define new parameters for the members of the step table. For example, if a parameter `'sample_parameter'` is defined as follows:

```
sample_parameter = protk_user_defined_function (list of arguments)
```

The definition suggests that the value of the parameter `sample_parameter` is set as a result of the calculation done using the function `protk_user_defined_function`.

The following steps are required to define new parameters using external relation functions:

1. Within the Creo Parametric TOOLKIT application, register an appropriate external function using `ProRelationFunctionRegister()`.
2. In the user interface for the step table, define the relation calling the externally registered functions.

Creo Parametric TOOLKIT relation functions called from an entry in the Step Table will have one of two owners for the relation set (`ProRelSet`) depending on the following cases:

Case 1: If the step has been applied

The owner will be the manufacturing feature created by this step table entry.

Case 2: If the step has not been applied

The owner will be an object whose type is `PRO_NC_STEP_OBJECT`.

Using external functions, it is possible to interact with an external database or application, in order to set the value of some parameters in the Process Table. Because relations may be reevaluated many times during a regeneration cycle, PTC recommends that the external functions contain some control that prevents reconnection to the external database on each invocation.

Note

If no Creo Parametric TOOLKIT application has registered the needed external function, any relation using the function cannot be evaluated and is shown as an error. However, when defining Global Relations, even if an undefined function is encountered, the relation will pass the validation, but, the relation will not be applied to the steps because the function is not found.

Creating Manufacturing Features

The creation of manufacturing features mirrors the creation of solid features in Creo Parametric TOOLKIT. All features at the very least must define a feature type. Certain manufacturing features also have a requirement that some "non-redefinable" elements must be defined.

With the exception of fixtures, all features are created in the manufacturing solid. However, fixtures are owned by the manufacturing assembly.

Like solid feature creation, manufacturing feature creation consists of several distinct steps:

1. Create the feature element tree.
2. Add nodes or subtrees to the feature tree.
3. Create a selection that represents the model in which to construct the feature.
4. Create the feature.

The following sections document only the first two steps for manufacturing features, because the actual process of feature creation is common to all.

Creating Fixtures

A fixture setup feature is one of the simplest manufacturing features and contains a maximum of four elements.

You should name a fixture setup feature. Optionally, you can define the following:

- The time required to perform the setup
- The identifiers of the fixturing components
- Associated comments

To Create an Element Tree for a Fixture Setup Feature

1. Allocate space for the tree using the following call:
   ```
   ProElementAlloc (PRO_E_FEATURE_TREE);
   ```
2. Define the feature type element (`PRO_E_FEATURE_TYPE`) to be an integer of value `PRO_FEAT_FIXSETUP`.
3. Define the name (`PRO_E_FEAT_NAME`) to be a wide string.
4. Optionally, add the setup time (`PRO_E_SETUP_TIME`) as a double.
5. Optionally, add the component identifiers of the fixturing models (`PRO_E_FIXT_COMPONENTS`).

Note

Because this is a multivalue element, you can add multiple (integer) values to the `PRO_E_FIXT_COMPONENT` element.

When the tree is complete, you can pass it (and a selection handle that represents the manufacturing assembly) to the function `ProFeatureCreate()`.

Creating Workcells

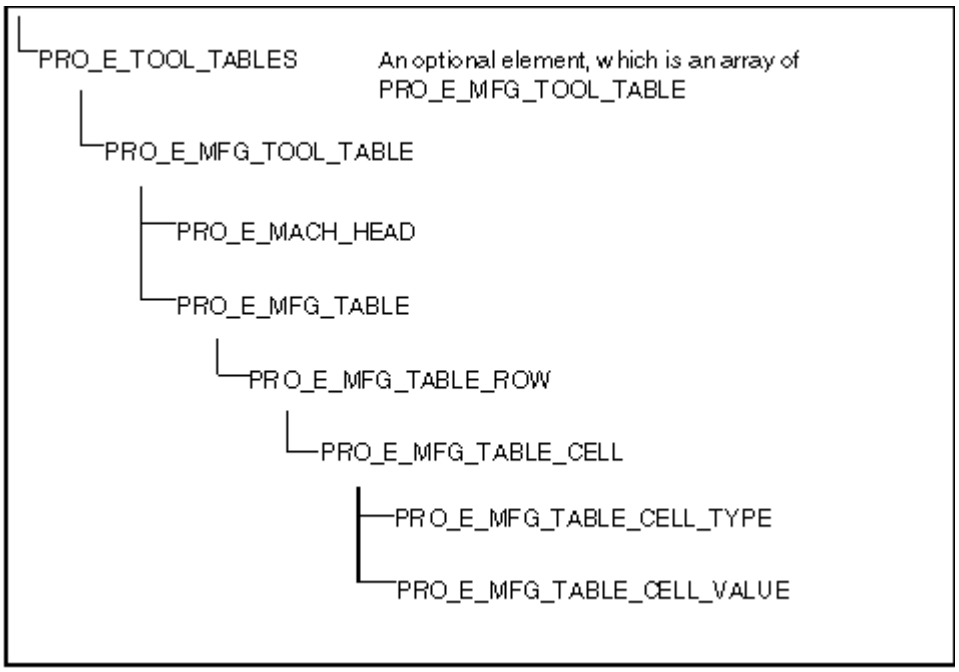
The element tree for workcells is described in the include file `ProWcell.h`. For this feature, the feature type element is `PRO_FEAT_WORKCELL`.

The following table shows the required and optional elements for workcell features. In this table, the "Value" column specifies whether the element is required (R) or optional (O).

Element	Description	Value
Cell type	Mill, mill/turn, and so on.	R
Number of axes	The number of axes.	R
Table direction	Horizontal or vertical (for turn or mill/turn)	R
Machine number heads	1 or 2 (for turn or mill/turn)	R
Name	The workcell name.	O
Tooling	Add tools to the workcell.	O
Tool table	Manipulate the tools in a tool table.	O
Parameters	The workcell parameters.	O

The feature element has two complex elements—`PRO_E_MFG_PARAMS`, described in the section Manufacturing Parameters on page 1448, and `PRO_E_TOOL_TABLE`. The following figure shows how the tool table element is constructed.

Tool Table Element



The first thing to note is that a workcell can have multiple tool tables, if it has more than one machine head. In this case, you can create a tool table for each head.

A manufacturing table is made up of an array of `PRO_E_MFG_TABLE_ROW` elements, which is itself an array of `PRO_E_MFG_TABLE_CELL` elements. Each `PRO_E_MFG_TABLE_CELL` is a compound element that contains two more elements—the cell type and its value.

The following examples show how to create a tool table with five drill bits, ranging in size from M8 to M16. The tool table for most workcells (excluding CMM) is of the following format:

```
POSITION   TOOL_ID   REGISTER  COMMENTS
1          drill_M8            8MM Drill
2          drill_M10           10MM Drill
```

For example, to create the first row of the table, you would define the following `PRO_E_MFG_TABLE_CELLs`:

- To define the first cell so the position of the tool is index 1, set the value of the `PRO_E_MFG_TABLE_CELL_TYPE` to `PRO_TOOL_TABLE_INDEX`, and set the value of `PRO_E_MFG_TABLE_CELL_VALUE` to the integer value 1.
- Similarly, define the tool identifier to be a `drill_M8`. Set the cell type element to `PRO_TOOL_TABLE_TOOL_ID`, and set the value to a wide string of value `drill_M8`. Because the register column is empty, it can be ignored. To set the comments element, set the type to `PRO_TOOL_TABLE_COMMENTS`, and set the value to a wide string of value “8MM Drill.”

Example 5: Creating a 2-Axis Lathe Workcell

The sample code in `UgMfgWcellCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to set up a simple 2-axis, horizontal lathe, using the previously created tool table. Note that both the `PRO_E_LATHE_DIR` and `PRO_E_MACH_NUM_HEADS` are required for this workcell type. All the other elements are optional.

Creating Operations

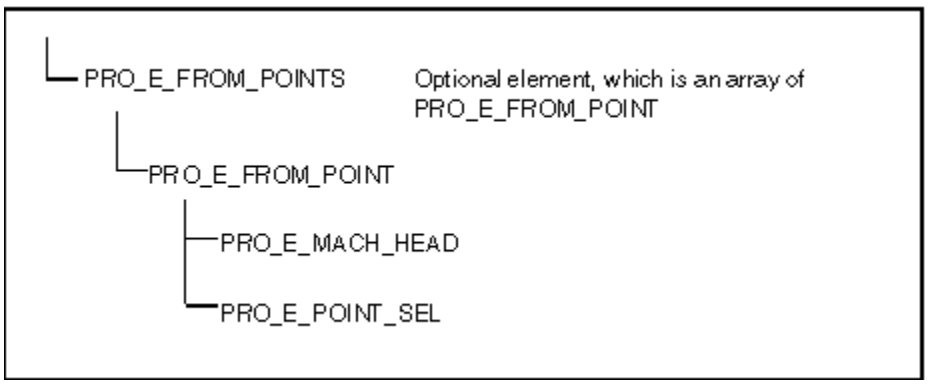
The element tree for manufacturing operations is described in the header file `ProMfgoper.h`. For this feature, the feature type element is `PRO_FEAT_WORKCELL`.

The following table shows the elements of a manufacturing operation. In this table, the "Value" column specifies whether the element is required (R) or optional (O).

Element	Description	Value
Workcell	The identifier of the workcell feature in which to perform the operation	R
Machine coordinate system	The identifier of the machining coordinate system	R
Name	The operation name	O
Comments	The operation comments	O
From point	The datum point from which to start the operation	O
Home point	The datum point on which to end the operation	O
Parameters	The operation parameters	O

The creation of the feature tree is simple, apart from the home and point elements. The following figure shows the element tree for the home points.

Home Point Element Tree



In this example, there is a from and home point defined for each machining head. If there is only one head, the value of the PRO_E_MACH_HEAD element should be 1. Note that the PRO_E_POINT_SEL selection should be initialized not to the datum point feature, but to the datum point geometry. To find this geometry, call the function ProFeatureGeomitemVisit()).

Example 6: Creating an Operation

The sample code in UgMfgOperCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg shows how to create an operation.

Creating NC Sequences

Header file ProNcseq.h describes the element tree for manufacturing NC sequences.

There are six supported NC sequence types:

- PRO_NCSEQ_PROF_SUR_MILL—Profile milling, feature type PRO_FEAT_MILL
- PRO_NCSEQ_VOL_MILL—Volume milling, feature type PRO_FEAT_MILL
- PRO_NCSEQ_CONV_SURF_MILL—Conventional surface, feature type PRO_FEAT_MILL
- PRO_NCSEQ_FACE_MILL—Face milling, feature type PRO_FEAT_MILL
- PRO_NCSEQ_PREV_TOOL_MILL—Local milling using previous tool, feature type PRO_FEAT_MILL
- PRO_NCSEQ_HOLEMAKING—Holemaking, type PRO_FEAT_DRILL

Like workcells, an NC sequence feature has a number of non-redefinable elements. For all NC sequences, the PRO_E_NCSEQ_TYPE and PRO_E_NUM_AXES elements are non-redefinable. For holemaking sequences, the PRO_E_HOLEMAKING_TYPE, PRO_E_PECK_TYPE, and PRO_E_HOLE_CYCLE_TYPE elements are non-redefinable. These elements are specific to holemaking sequences and need not be defined for milling sequences. The following table shows the elements of an NC sequence. In this table, the Value column specifies whether the element is required (R) or optional (O).

Element	Description	Value
Feature type	The feature type	R
Type	The sequence type	R
Operation	Operation to which to add the NC sequence	R
Retraction plane	The retraction plane	R
Tool	The tool	R
Csys	The manufacturing coordinate system	R

Element	Description	Value
Parameters	The manufacturing parameters	R
Name	The name of the NC sequence	O
Number of axes	The number of axes	O
Machine head	The machine head	O
Fixture	The fixture	O
Entities to be machined		
Surface	The surface	R
Holes or volume	The holes or volume	R
Start path	The start path	R
End path	The end path	R

The retraction plane that must be defined as part of the NC sequence requires the identifier of the underlying geometry of the datum plane. To obtain this identifier, visit the datum plane geometry items using `ProFeatureGeomitemVisit()`.

There are also a number of required parameters for each NC sequence that must be defined. For conventional milling, the following parameters are required:

- `CUT_FEED`
- `TOLERANCE`
- `STEP_OVER`
- `SPINDLE_SPEED`
- `CLEAR_DIST`

For face milling, the following parameters are required:

- `CUT_FEED`
- `STEP_DEPTH`
- `TOLERANCE`
- `STEP_OVER`
- `SPINDLE_SPEED`
- `CLEAR_DIST`

For holemaking, the following parameters are required:

- `CUT_FEED`
- `TOLERANCE`
- `SPINDLE_SPEED`
- `CLEAR_DIST`

Both milling and holemaking features elements require that the entities (and some associated properties) to be machined are set by API functions, rather than by element tree. Like the standard elements, these functions require a call to `ProElementAlloc()` to reserve space for the elements. Once the elements are complete, you can add them to the tree like the other standard elements.

Note

Currently, using `ProFeatureElemtreeExtract()` with NC sequences yields an element tree without holes or surface elements. In other words, there is no way to retrieve hole set or surface information.

Milling-Specific Functions

Functions Introduced:

- **ProNcseqElemSurfaceAdd()**
- **ProNcseqElemMillsurfSet()**
- **ProNcseqElemSurfaceflipSet()**

After you allocate the surface element with `ProElementAlloc()`, you can add the surface to be milled to the element using the function `ProNcseqElemSurfaceAdd()`.

If the model contains a milling surface, you can set the whole surface in the NC sequence using `ProNcseqElemMillsurfSet()`. To control its orientation, call `ProNcseqElemSurfaceflipSet()`.

Example 7: Adding Surfaces

The sample code in `UgMfgSrfAdd.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to use the function `ProNcseqElemSurfaceAdd()`.

Holemaking-Specific Functions

To add hole sets to an element tree, first, obtain the hole set number by calling the function `ProNcseqElemHolesetAdd()`. This hole set is used to reference a set of holes with the same properties, including depth, direction, countersink diameter, and so on. Note that feature element `PRO_E_HOLESETS` has limited support for drill point sets, but allows the user to identify and to delete existing drill point sets in a feature, or to overwrite drill point sets with drill axes sets.

For example, you might want one set of holes to be countersunk to a diameter of 10mm, and another set to 14mm. Because these countersinking operations are done with the same tool, they should be in the same NC sequence. Because the countersink diameter is different for each, you should create two hole sets.

Note

The following functions (and the element tree `PRO_E_HOLES`) exist in Pro/TOOLKIT Revisions 20 and later only to provide backwards compatibility. Use the more complete and powerful element tree `PRO_E_HOLESETS` for holemaking functions.

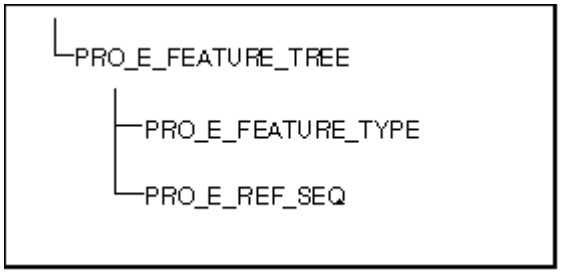
- `ProNcseqElemHolesetAdd()`
- `ProNcseqElemHolesetDepthTypeSet()`
- `ProNcseqElemHolesetDepthBySet()`
- `ProNcseqElemHolesetDepthSet()`
- `ProNcseqElemHolesetStartSet()`
- `ProNcseqElemHolesetEndSet()`
- `ProNcseqElemHolesetDirectionSet()`
- `ProNcseqElemHolesetAxisAdd()`
- `ProNcseqElemHolesetDrillpartAdd()`
- `ProNcseqElemHolesetCsinkdiamSet()`

Creating Material Removal Volumes

In Creo Parametric, material removal features can be created by defining geometry to represent the volume removed, or they can be calculated automatically from the NC sequence. The current release of Creo Parametric TOOLKIT supports automatic material removal feature creation only.

The feature tree is very simple, as shown in the following figure.

Feature Tree



Set the PRO_E_FEATURE_TYPE value to PRO_FEAT_MAT_REMOVAL, and set the PRO_E_REF_SEQ to the identifier of the NC sequence feature from which to create the material removal volume.

Example 8: Creating a Conventional Milling Sequence

The sample code in UgMfgMillSeqCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg shows how to create a milling feature using an element tree.

Analyzing Manufacturing Features

Functions Introduced:

- **ProMfgoperToolpathDisplay()**
- **ProNcseqToolpathDisplay()**
- **ProNcseqNumGet()**
- **ProNcseqCutTimeGet()**
- **ProNcseqRemovedVolGet()**

You can use the functions ProMfgoperToolpathDisplay() and ProNcseqToolpathDisplay() to invoke the corresponding toolpath for the specified object. The function ProNcseqNumGet() returns the number of the specified NC sequence.

The final two functions access the machining time and the volume of material removed during the machining sequence.

62

Production Applications: Customized Tool Database

Creo Parametric TOOLKIT supports integration of theCreo NC tool search command with external tool databases. These functions and callbacks allow users to create queries for third-party tool manager applications. These applications query external databases (you can specify more than one) and return logical tool data to Creo Parametric.

Overview

Creo Parametric TOOLKIT supplies functions that allow integration of the Tool Selection user interface with a third-party tool manager application. This integration consists of includes three areas of functionality:

- The ability to preregister one or more databases with the Creo Parametric user interface, and to assign custom search parameters to each database. Custom search parameters are items which would not be included natively inside of Creo Parametric, but which could be used to initiate or narrow the tool search.
- Callback functions, that are invoked when the user opts to search inside the tool database. Within the callback functions, the application can extract the native and non-negative search parameters and use these parameters to execute the search.
- The ability to return a list of detailed results to Creo Parametric to be displayed in the results dialog box.

Setting up the Database and Custom Search Parameters

The functions described in this section enable you to setup the external database and custom search parameters supported by the database.

Functions Introduced:

- **ProMfgdbDataAlloc()**
- **ProMfgdbDataDbnameAdd()**
- **ProMfgdbNameCreate()**
- **ProMfgdbSearchoptCreate()**
- **ProMfgdbDataSearchoptAdd()**
- **ProMfgdbSearchoptApplicDataAdd()**
- **ProMfgdbSearchoptAllowedValueAdd()**

The function `ProMfgdbDataAlloc()` allocates a `ProMfgdbData` handle containing database and search information.

The function `ProMfgdbDataDbnameAdd()` adds the name of an external tool database to the handle allocated by `ProMfgdbDataAlloc()`. To specify more than one database, make multiple calls to `ProMfgdbDataDbnameAdd()`.

The function `ProMfgdbNameCreate()` allocates memory for a handle containing the name of a group into which search options are organized.

The function ProMfgdbSearchoptCreate() allocates and initializes a structure for a custom search option. It requires that the option be assigned to a group, typically this would be allocated from ProMfgdbNameCreate().

The function ProMfgdbDataSearchoptAdd() adds a custom search option to the ProMfgdbData handle.

The function ProMfgdbSearchoptApplicDataAdd() specifies the category and object type for which a custom search option is valid. Call this function at least once for each option. Assign an option to multiple categories or object types with multiple calls to this function.

The function ProMfgdbSearchoptAllowedValueAdd() adds valid values available for a search option. Assign multiple values with multiple calls to this function.

Registering the External Database

The function described in this section allows you to register the Creo Parametric TOOLKIT application to search the external tool database.

Function Introduced:

- **ProMfgdbRegister()**

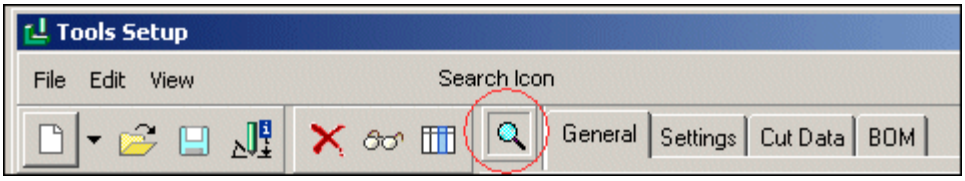
The function ProMfgdbRegister() registers an external database with Creo Parametric. You should supply the ProMfgdbData handle you created for your external database(s).

The function also requires that you supply three callbacks:

- A ProMfgdbLoginAction, which is called by Creo Parametric when the user attempts to initiate access to the external database.
- A ProMfgdbLogoffAction, which is called when the user is ready to close the connection to the external database.
- A ProMfgdbSearchAction, which provides the custom implementation for the search, and provides the results back to Creo Parametric.

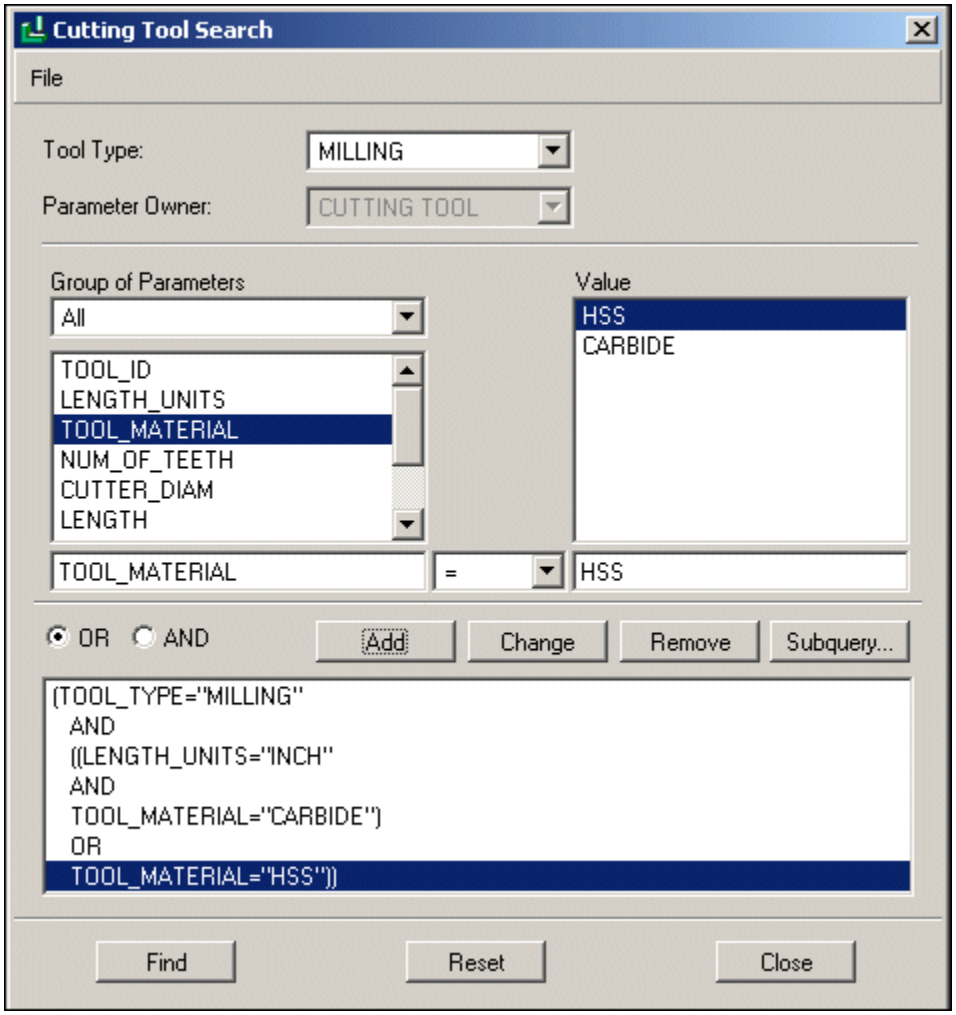
Finally, the function requires that you supply the object category. Although there are several categories listed for this function, currently the function supports only the following category:

- PROMFGDBCAT_CUTTING_TOOL
    - When one or more external databases have been registered with Creo Parametric, a search icon is displayed in the **Tool Setup** dialog box in Creo Parametric as shown in the following figure. Click the search icon to invoke the **Cutting Tool Search** dialog box.



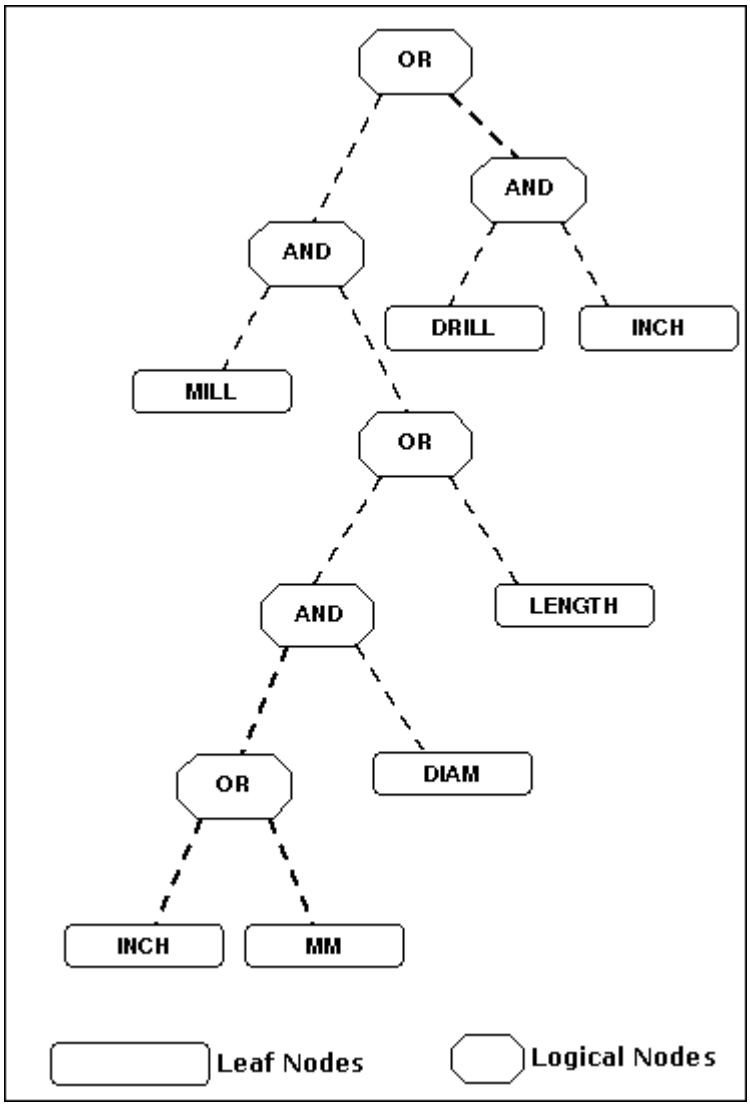
Querying the External Database

When users invoke the Search icon to construct a query, they will be presented with the Cutting Tool Search dialog, as shown in the following figure. Within the dialog, users can specify the query parameters, values and group the query constraints with logical operators. When the user completes the query and executes the Find command, Creo Parametric will call the Creo Parametric TOOLKIT application with the data from this dialog.



The functions described in this section enable you to convert the Creo Parametric queries into a format that can be used with the external database.

Creo Parametric tool database queries are arranged in a tree format, as shown in the following figure. Leaf nodes contain expressions, for example, units and type.



Functions Introduced:

- **ProMfgdbQuerynodeIsLeaf()**
- **ProMfgdbQuerynodeLeftChildGet()**
- **ProMfgdbQuerynodeRightChildGet()**
- **ProMfgdbQuerynodeLogicOperGet()**
- **ProMfgdbQuerynodeExprGet()**
- **ProMfgdbQueryTargetGet()**

- **ProMfgdbExprNameGet()**
- **ProMfgdbExprCategoryGet()**
- **ProMfgdbExprValueGet()**
- **ProMfgdbExprValueTypeGet()**
- **ProMfgdbExprCompopGet()**

`ProMfgdbQuerynodeIsLeaf()`

The functions `ProMfgdbQuerynodeLeftChildGet()` and `ProMfgdbQuerynodeRightChildGet()` return either the left or right branch of a query node, respectively.

`ProMfgdbQuerynodeLogicOperGet()ProMfgdbQuerynodeIsLeaf()`

`ProMfgdbQuerynodeExprGet()`

`ProMfgdbQueryTargetGet()`

After the query functions return expressions from a leaf node, your Creo Parametric TOOLKIT application can gather information contained in the expressions. The following functions return attributes and operators contained in the returned expressions of a leaf node.

The function `ProMfgdbExprNameGet()` returns the name of the attribute contained in the specified expression. The function `ProMfgdbExprCategoryGet()` returns the category of the attribute contained in the specified expression. The functions `ProMfgdbExprValueGet()` and `ProMfgdbExprValueTypeGet()` return the value and value type contained in the specified expression, respectively. The function `ProMfgdbExprCompopGet()` returns the comparison operator (=, <, >, and so on) contained in the specified expression.

Returning the Search Results

The functions in this section enable you to populate the results of the query to the external database and pass them back to Creo Parametric so that Creo Parametric will display them appropriately in the results window.

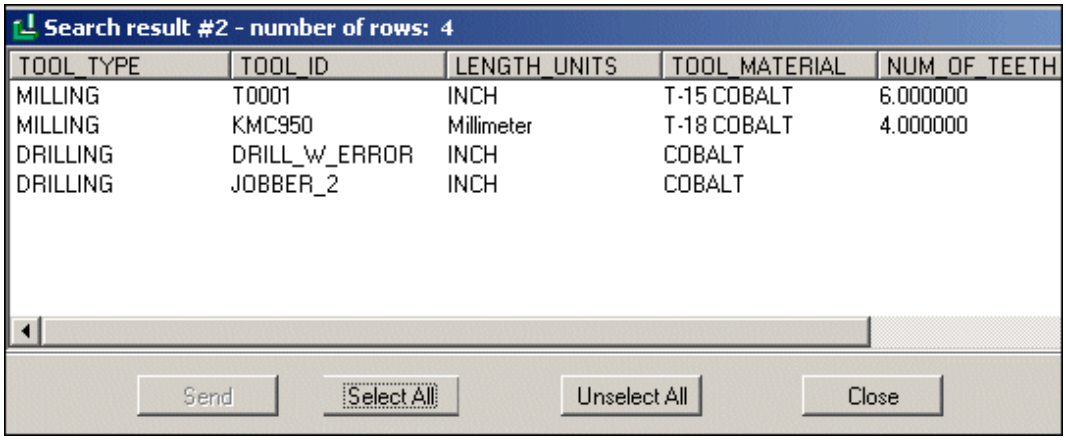
Functions Introduced:

- **ProMfgdbMatchAlloc()**
- **ProMfgdbMatchParamAdd()**

Use the function `ProMfgdbMatchAlloc()` to allocate memory for the structure used to store a match to the query.

The function `ProMfgdbMatchParamAdd()` adds a parameter to the match structure. Make multiple calls to this function to add multiple parameters to the match.

An example of the results passed back from a database query is shown in the following figure:



63

Production Applications: Creo NC Sequences, Operations and Work Centers

This chapter describes the Creo Parametric TOOLKIT support for Creo NC sequences.

Overview

This section describes the Creo Parametric TOOLKIT functions that enable you to access the following types of manufacturing features of Creo NC Sequences, Operations and Work Centers.

Creo NC Step	Header File
Roughing	ProMfgFeatRoughing.h
Reroughing	ProMfgFeatReroughing.h
Finishing	ProMfgFeatFinishing.h
Corner Finishing	ProMfgFeatCornerFinishing.h
2-Axis Curve Trajectory Milling	ProMfgFeat2xCurvTraj.h
3-Axis Trajectory Milling	ProMfgFeatTrajectory.h
Profile Milling	ProMfgFeatProfMilling.h
Manual Cycle	ProMfgFeatManualCycle.h
Thread Milling	ProMfgFeatThreadMilling.h
Area Turning	ProMfgFeatTurning.h
Groove Turning	ProMfgFeatTurning.h
Profile Turning	ProMfgFeatTurning.h
Face Milling	ProMfgFeatFacing.h
Creo NC Operation Definition	ProMfgFeatOperation.h
Workcell Definition	ProMfgFeatWcellWedm.h ProMfgFeatWcellMill.h ProMfgFeatWcellMillTurn.h ProMfgFeatWcellLathe.h ProMfgFeatWcellCmm.h
Fixture Definition	ProMfgFeatFixture.h
Thread Turning	ProMfgFeatTurnThread.h
Holemaking	ProMfgFeatHolemaking.h
Shut-off Surface Feature	ProMoldShutSrf.h
Round and Chamfer	ProMfgFeatRoundChamferMilling.h
Engraving	ProMfgFeatEngraving.h
Cutline Milling	ProMfgFeatCutlineMilling.h
Drill Group Feature	ProMfgFeatDrillGroup.h

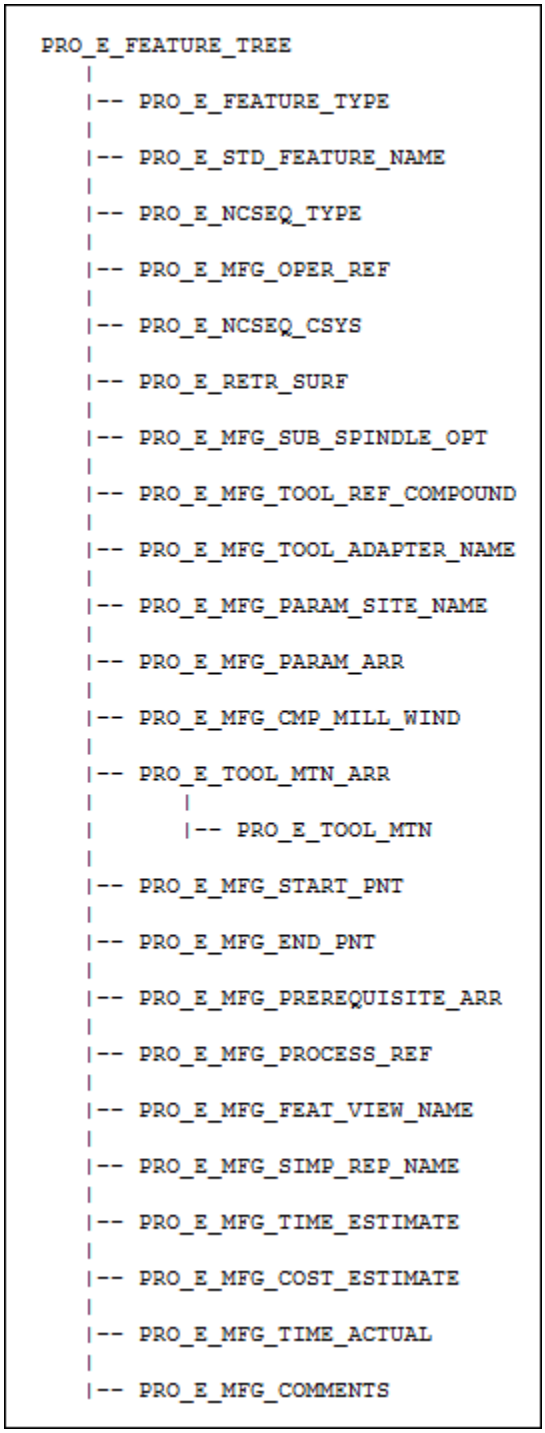
Element Trees: Roughing Step

This section describes how to construct and access the element tree for a milling roughing feature. It also describes how to create, redefine, and access the properties of these features.

The Roughing Feature Element Tree:

The element tree for the milling roughing sequence is documented in the header file `ProMfgFeatRoughing.h`, and is as shown in the following figure:

Element Tree for Roughing feature



The following table describes the elements in the element tree for the Roughing feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Roughing_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_NCSEQ_ROUGHVOL_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters.

Element ID	Data Type	Description
		Note The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_CMP_MILL_WIND	Compound	Mandatory compound element. Specifies the mill window compound definition. For more information, refer to the section Surface Collection with Mill Window on page 1680.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion type for this element is : PRO_TM_TYPE_AUTOMATIC_CUT. For more information, refer to the section Tool Motion — Auto Cut on page 1766.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.

Element ID	Data Type	Description
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

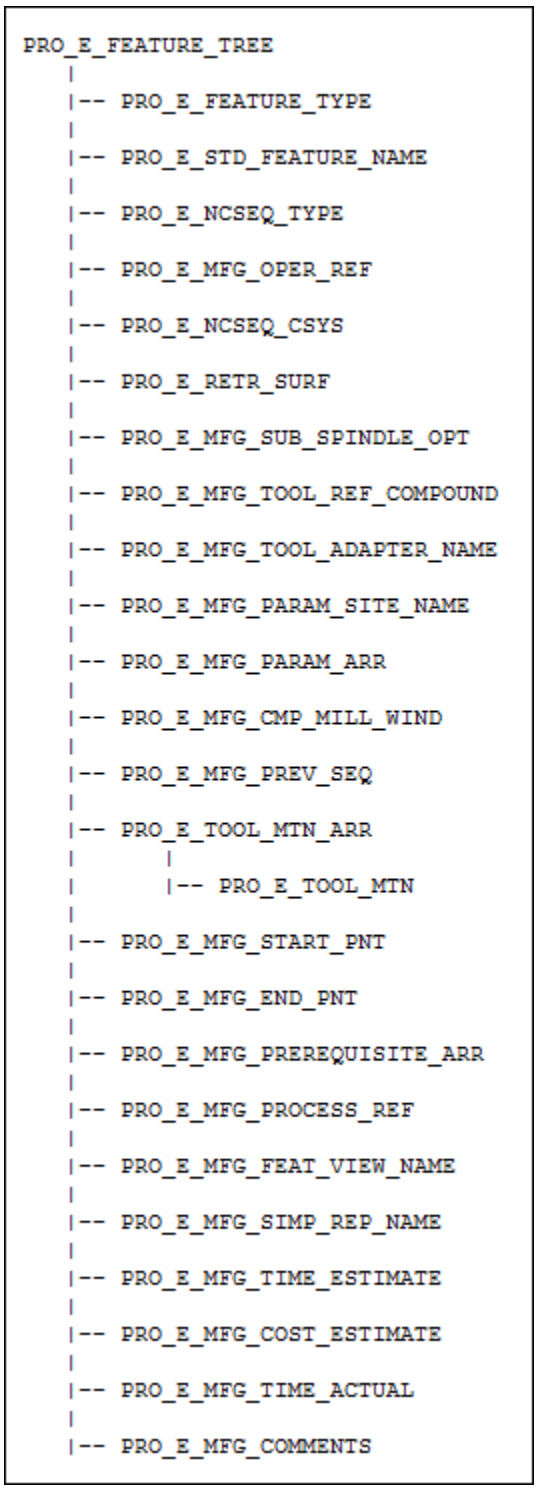
Element Trees: Reroughing Step

This section describes how to construct and access the element tree for a milling reroughing feature. It also describes how to create, redefine, and access the properties of these features.

The Reroughing Feature Element Tree:

The element tree for the milling reroughing sequence is documented in the header file `ProMfgFeatReroughing.h`, and is as shown in the following figure:

Element Tree for Reroughing feature



The following table describes the elements in the element tree for the Reroughing feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature . The valid value for this element isPRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence feature name. The default value is Re-roughing_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence type. The valid value for this element is PRO_NCSEQ_REROUGH_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters.

Element ID	Data Type	Description
		Note The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_CMP_MILL_WIND	Compound	Mandatory compound element. Specifies the mill window compound definition. For more information, refer to the section Surface Collection with Mill Window on page 1680.
PRO_E_MFG_PREV_SEQ	PRO_VALUE_TYPE_ SELECTION	Mandatory Element. Specifies the selection sequence feature that requires removal of left over material. The valid values for this element are: • Volume Milling • Profile Milling • Re-roughing sequence
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion type for this element is PRO_TM_TYPE_ AUTOMATIC_CUT. For more information, refer to the section Tool Motion — Auto Cut on page 1766 .
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_	Optional element. Specifies the

Element ID	Data Type	Description
	SELECTION	reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step.. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

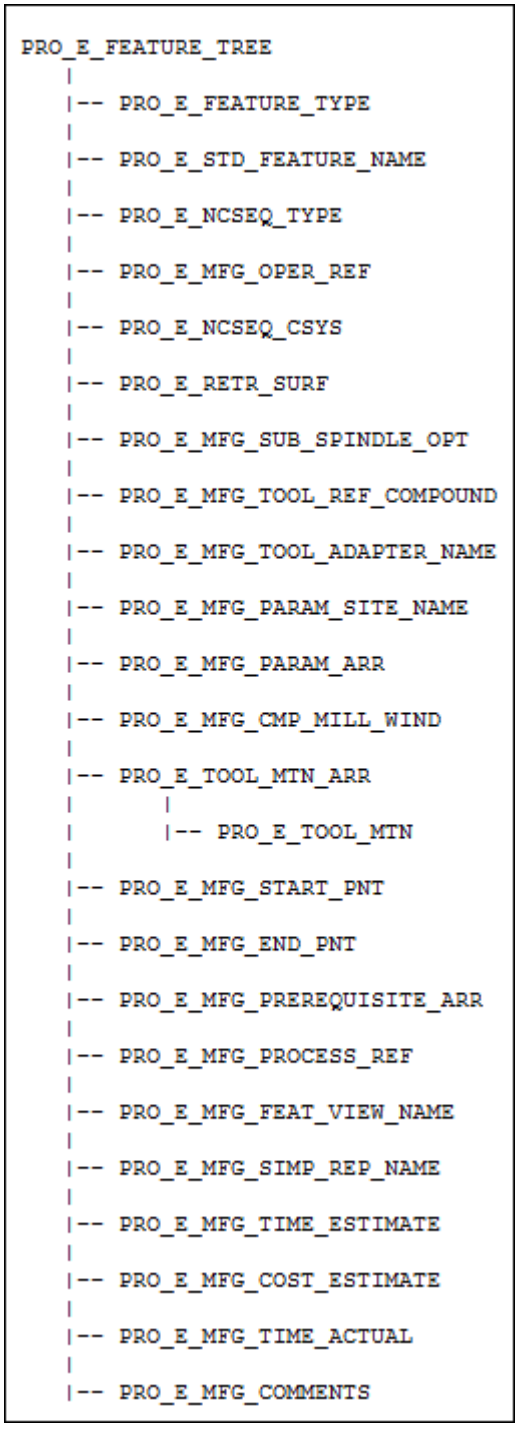
Element Trees: Finishing Step

This section describes how to construct and access the element tree for milling Finishing feature. It also describes how to create, redefine, and access the properties of these features.

The Finishing Feature Element Tree:

The element tree for the milling Finishing sequence is documented in the header file `ProMfgFeatFinishing.h`, and is as shown in the following figure:

Element Tree for Finishing feature



The following table describes the elements in the element tree for the Finishing feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Finishing_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_NCSEQ_CVNC_FINISH_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters.

Element ID	Data Type	Description
		Note The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_CMP_MILL_WIND	Compound	Mandatory compound element. Specifies the mill window compound definition. For more information, refer to the section Surface Collection with Mill Window on page 1680.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion type for this element is : PRO_TM_TYPE_AUTOMATIC_CUT. For more information, refer to the sectionTool Motion — Auto Cut on page 1766 .
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682 .
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.

Element ID	Data Type	Description
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: Corner Finishing Step

This section describes how to construct and access the element tree for a corner finishing step. It also describes how to create, redefine, and access the properties of these features.

The Corner Finishing Element Tree:

The element tree for the corner finishing sequence is documented in the header file `ProMfgFeatCornerFinishing.h`, and is as shown in the following figure:

Element Tree for Corner Finishing Step

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_NCSEQ_TYPE
    |
    |-- PRO_E_MFG_OPER_REF
    |
    |-- PRO_E_NCSEQ_CSYS
    |
    |-- PRO_E_RETR_SURF
    |
    |-- PRO_E_MFG_SUB_SPINDLE_OPT
    |
    |-- PRO_E_MFG_TOOL_REF_COMPOUND
    |
    |-- PRO_E_MFG_TOOL_ADAPTER_NAME
    |
    |-- PRO_E_MFG_PARAM_SITE_NAME
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_MFG_CMP_MILL_WIND
    |
    |-- PRO_E_MFG_PREV_TOOL_ID
    |
    |-- PRO_E_TOOL_MTN_ARR
    |       |
    |       |-- PRO_E_TOOL_MTN
    |
    |-- PRO_E_MFG_START_PNT
    |
    |-- PRO_E_MFG_END_PNT
    |
    |-- PRO_E_MFG_PREREQUISITE_ARR
    |
    |-- PRO_E_MFG_PROCESS_REF
    |
    |-- PRO_E_MFG_FEAT_VIEW_NAME
    |
    |-- PRO_E_MFG_SIMP_REP_NAME
    |
    |-- PRO_E_MFG_TIME_ESTIMATE
    |
    |-- PRO_E_MFG_COST_ESTIMATE
    |
    |-- PRO_E_MFG_TIME_ACTUAL
    |
    |-- PRO_E_MFG_COMMENTS
```

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Corner_Finishing_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_NCSEQ_CVNC_CORN_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters.

Element ID	Data Type	Description
		Note The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_CMP_MILL_WIND	Compound	Mandatory compound element. Specifies the mill window compound definition. For more information, refer to the section Surface Collection with Mill Window on page 1680.
PRO_E_MFG_PREV_TOOL_ID	PRO_VALUE_TYPE_WSTRING	Optional element. Name of cutting tool (tool id) which will be used for calculating the remainder material.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion type for this element is : PRO_TM_TYPE_AUTOMATIC_CUT. For more information, refer to the section Tool Motion — Auto Cut on page 1766 .
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682 .
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to

Element ID	Data Type	Description
		be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: 3–Axis Trajectory Milling Step

This section describes how to construct and access the element tree for the 3 Axis Trajectory Milling feature. It also describes how to create, redefine, and access the properties of these features. For 2-axis curve trajectory milling step, refer to the section Manufacturing 2–Axis Curve Trajectory Milling Step on page 1496.

The element tree for the 3 axis trajectory milling sequence is documented in the header file `ProMfgFeatTrajectory.h`, and is as shown in the following figure:

Element Tree for 3 Axis Trajectory Milling feature

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_NCSEQ_TYPE
    |
    |-- PRO_E_MFG_OPER_REF
    |
    |-- PRO_E_MFG_SEQ_NUM_AXES_OPT
    |
    |-- PRO_E_NCSEQ_CSYS
    |
    |-- PRO_E_RETR_SURF
    |
    |-- PRO_E_MFG_SUB_SPINDLE_OPT
    |
    |-- PRO_E_MFG_TOOL_REF_COMPOUND
    |
    |-- PRO_E_MFG_TOOL_ADAPTER_NAME
    |
    |-- PRO_E_MFG_PARAM_SITE_NAME
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_CHECK_SURF_COLL
    |
    |-- PRO_E_TOOL_MTN_ARR
    |       |
    |       |-- PRO_E_TOOL_MTN
    |
    |-- PRO_E_MFG_START_PNT
    |
    |-- PRO_E_MFG_END_PNT
    |
    |-- PRO_E_MFG_PREREQUISITE_ARR
    |
    |-- PRO_E_MFG_PROCESS_REF
    |
    |-- PRO_E_MFG_FEAT_VIEW_NAME
    |
    |-- PRO_E_MFG_SIMP_REP_NAME
    |
    |-- PRO_E_MFG_TIME_ESTIMATE
    |
    |-- PRO_E_MFG_COST_ESTIMATE
    |
    |-- PRO_E_MFG_TIME_ACTUAL
    |
    |-- PRO_E_MFG_COMMENTS
```

The following table describes the elements in the element tree for the 3 Axis Trajectory Milling feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature . The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence . The default value is Trajectory_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence type. The valid value for this element isPRO_NCSEQ_TRAJ_MILL_STEP.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_MFG_SEQ_NUM_AXES_OPT	PRO_VALUE_TYPE_INT	Specifies the number of axes for milling. You can specify 3, 4, or 5 axes. **Note** You can set this element to 5 only if the work center allows 5-axis machining.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673 .
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676 .

Element ID	Data Type	Description
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677 .
PRO_E_CHECK_SURF_COLL	PRO_VALUE_TYPE_SELECTION	Optional compound element. Specifies the check surfaces compound definition. For more information, refer to the section Checking Surfaces on page 1687.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion types for this element are : • PRO_TM_TYPE_CURVE_TRAJECTORY. For more information, refer to the section Tool Motion — Curve Trajectory on page 1740. • PRO_TM_TYPE_SURF_TRAJECTORY. For more information, refer to the section Tool Motion — Surface Trajectory on page 1751. • PRO_TM_TYPE_FOLLOW_CUT. For more information, refer to the section Tool Motion — Follow Cut on page 1770. • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO

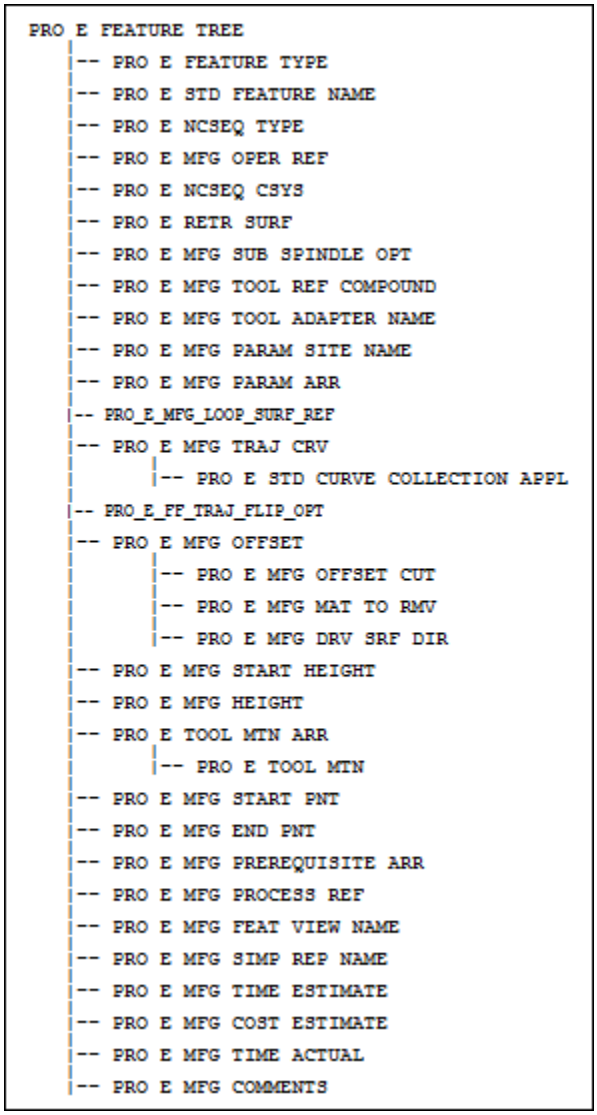
Element ID	Data Type	Description
		_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715.

Element ID	Data Type	Description
		• PRO_TM_TYPE_HELICAL_APPROACH. For more information, refer to the section Tool Motion — Helical Approach on page 1717. • PRO_TM_TYPE_HELICAL_EXITFor more information, refer to the section Tool Motion — Helical Exit on page 1720. • PRO_TM_TYPE_RAMP_EXIT. For more information, refer to the section Tool Motion — Ramp Exit on page 1760. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762. • PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682 .
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step.

Element ID	Data Type	Description
		Note This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Manufacturing 2–Axis Curve Trajectory Milling Step

The element tree for the 2–axis curve trajectory sequence is documented in the header file `ProMfgFeat2xCurvTraj.h`, and is as shown in the following figure:



The following table describes the elements in the element tree for the 2–axis curve trajectory sequence feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature . The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the sequence feature name. The default value is Curve_Trajectory_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence type. The valid value for this element isPRO_NCSEQ_FF_TRAJ_

Element ID	Data Type	Description
		MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the coordinate system datum feature to be used as a sequence coordinate system.
PRO_E_RETR_SURF	Compound	Mandatory element. Specifies the retract compound definition. The element tree for the Retract Elements is defined in the section Retract Elements on page 1673. For more information, refer to the section Retract Elements for more information on the element tree.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory element. Specifies the tool reference compound definition. The element tree for the Tool Reference is defined in the header file ProMfgElemToolRef.h. For more information, refer to the section Tool Reference on page 1676for more information on the element tree.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of site with default values for manufacturing parameters. **Note** The name is going to be ignored if site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing

Element ID	Data Type	Description
		parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file `ProMfgElemParam.h`. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
`PRO_E_MFG_LOOP_SURF_REF`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Select the surfaces for which the loops of edges must be collected. The collected loops of edges are machined.
`PRO_E_MFG_TRAJ_CRV`	`Compound`	Mandatory element. Specifies the machining curves compound definition.
`PRO_E_STD_CURVE_COLLECTION_APPL`	`Curve Collection`	Mandatory element. Specifies the curve collection.
`PRO_E_FF_TRAJ_FLIP_OPT`	`PRO_VALUE_TYPE_INT`	Optional element. Flips the machining direction of the tool. The valid values for this element are: • `PRO_B_TRUE`—Specifies that the default direction on the curve will be used. • `PRO_B_FALSE`—Specifies that the opposite direction on the curve will be used.
`PRO_E_MFG_OFFSET`	`Compound`	Optional element. Specifies the offset compound definition.
`PRO_E_MFG_OFFSET_CUT`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the offset cut. The valid values for this element are: • `PRO_B_TRUE`—Tool offset will be applied. • `PRO_B_FALSE`—Tool offset will not be applied.
`PRO_E_MFG_MAT_TO_RMV`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the material side. The valid values for this element are: • `PRO_MFG_DIR_SAME`—Specifies the default side will be used. • `PRO_MFG_DIR_OPPOSITE`—Specifies that the default side will be flipped.
`PRO_E_MFG_DRV_SRF_DIR`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the flip drive Surface direction. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_B_FALSE—Specifies that the default direction on the drive surface will be used. • PRO_B_TRUE—Specifies that the opposite direction on the drive surface will be used.
PRO_E_MFG_START_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of the plane where machining will begin.
PRO_E_MFG_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies single selection of the plane for the tool tip to follow.
PRO_E_TOOL_MTN_ARR	Array	Optional element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Optional element. Specifies the tool motion compound specification.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to start machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to end machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. Specifies the array of ids of prerequisite sequences. The element tree for the Sequence Prerequisites is defined in the header file ProMfgElemPrerequisite.h. For more information, refer to the section Sequence Prerequisites on page 1682 for more information on the element tree.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometry references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the view name. It allows you to associate specific view with a machining step.

Element ID	Data Type	Description
		Note This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. It allows you to associate a specific simplified representation with a machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimate. It allows you to specify time estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate. It allows you to specify cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time. It allows you to specify actual time for the machining step **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: Manual Cycle Step

This section describes how to construct and access the element tree for manual milling cycle feature. It also describes how to create, redefine, and access the properties of these features.

The Manual Cycle Step Feature Element Tree:

The element tree for the manual cycle step feature is documented in the header file `ProMfgFeatManualCycle.h`, and is as shown in the following figure:

Element Tree for Manual Cycle Step feature

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_NCSEQ_TYPE
    |
    |-- PRO_E_MFG_OPER_REF
    |
    |-- PRO_E_NCSEQ_CSYS
    |
    |-- PRO_E_RETR_SURF
    |
    |-- PRO_E_MFG_SUB_SPINDLE_OPT
    |
    |-- PRO_E_MFG_TOOL_REF_COMPOUND
    |
    |-- PRO_E_MFG_TOOL_ADAPTER_NAME
    |
    |-- PRO_E_MFG_PARAM_SITE_NAME
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_TOOL_MTN_ARR
    |       |
    |       |-- PRO_E_TOOL_MTN
    |
    |-- PRO_E_MFG_START_PNT
    |
    |-- PRO_E_MFG_END_PNT
    |
    |-- PRO_E_MFG_PREREQUISITE_ARR
    |
    |-- PRO_E_MFG_PROCESS_REF
    |
    |-- PRO_E_MFG_FEAT_VIEW_NAME
    |
    |-- PRO_E_MFG_SIMP_REP_NAME
    |
    |-- PRO_E_MFG_TIME_ESTIMATE
    |
    |-- PRO_E_MFG_COST_ESTIMATE
    |
    |-- PRO_E_MFG_TIME_ACTUAL
    |
    |-- PRO_E_MFG_COMMENTS
```

The following table lists the contents of `PRO_E_TOOL_MTN` element.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature . The valid values for this element are: • PRO_FEAT_MILL— For milling manual cycle. • PRO_FEAT_TURN— For turning manual cycle.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Manual_Cycle_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid values for this element are: • PRO_SEQ_MANUAL_CYCLE_MILL— For milling manual cycle. • PRO_SEQ_MANUAL_CYCLE_TURN— For turning manual cycle.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_Creo NC SEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	This compound element specifies retract definition. For more information, refer to the section Retract Elements on page 1673 **Note** This element is mandatory for milling manual Creo NC sequences and is ignored for turning manual sequences. .
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page

Element ID	Data Type	Description
		1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name. **Note** This element is optional for milling manual sequences and is ignored for turning manual sequences.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion types for this element are: • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700.

Element ID	Data Type	Description
		• PRO_TM_TYPE_GOTO_SURFACE— This tool motion type is for milling manual cycle only. For more information, refer to the section Tool Motion — Go To Surface on page 1722. • PRO_TM_TYPE_GOTO_AXIS — This tool motion type is for milling manual cycle only. For more information, refer to the section Tool Motion — Go To Axis on page 1724. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_PLUNGE— This tool motion type is for milling manual cycle only. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT— This tool motion type is for milling manual cycle only. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767. The elements related to tool motion are defined in ProMfgElemToolMtnAutoCut.h. For more information, refer to the section Tool Motion — Auto Cut on page 1766 .
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to

Element ID	Data Type	Description
		end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.

Element ID	Data Type	Description
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: Thread Milling

This section describes how to construct and access the element tree for a thread roughing feature. It also describes how to create, redefine, and access the properties of these features.

The Thread Milling Feature Element Tree:

The element tree for the thread milling Creo NC sequence is documented in the header file `ProMfgFeatThreadMilling.h`, and is as shown in the following figure:

Element tree for PRO_E_TOOL_MTN element

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_NCSEQ_TYPE
    |
    |-- PRO_E_MFG_OPER_REF
    |
    |-- PRO_E_NCSEQ_CSYS
    |
    |-- PRO_E_RETR_SURF
    |
    |-- PRO_E_MFG_MILL_THREAD_TYPE_OPT
    |
    |-- PRO_E_MFG_MILL_THREAD_TAPER_OPT
    |
    |-- PRO_E_MFG_SUB_SPINDLE_OPT
    |
    |-- PRO_E_MFG_TOOL_REF_COMPOUND
    |
    |-- PRO_E_MFG_TOOL_ADAPTER_NAME
    |
    |-- PRO_E_MFG_PARAM_SITE_NAME
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_MFG_THRM_HOLESET_ARR
    |
    |-- PRO_E_TOOL_MTN_ARR
    |       |
    |       |-- PRO_E_TOOL_MTN
    |
    |-- PRO_E_MFG_START_PNT
    |
    |-- PRO_E_MFG_END_PNT
    |
    |-- PRO_E_MFG_PREREQUISITE_ARR
    |
    |-- PRO_E_MFG_PROCESS_REF
    |
    |-- PRO_E_MFG_FEAT_VIEW_NAME
    |
    |-- PRO_E_MFG_SIMP_REP_NAME
    |
    |-- PRO_E_MFG_TIME_ESTIMATE
    |
    |-- PRO_E_MFG_COST_ESTIMATE
    |
    |-- PRO_E_MFG_TIME_ACTUAL
    |
    |-- PRO_E_MFG_COMMENTS
```

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Thread_Milling_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_NCSEQ_THREAD_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673. **Note** This element is mandatory when PRO_E_DRILL_MODE is set to PRO_DRILL_HOLE_ON_MILL and is ignored when set to PRO_DRILL_HOLE_ON_LATHE.
PRO_E_MFG_MILL_THREAD_TYPE_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the thread type. The valid values for this element are: • External • Internal The value for this element is defined by the ProMillThreadType parameter.
PRO_E_MFG_MILL_THREAD_TAPER_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the taper type. The valid values for this element are: • None • NTP • Custom The value for this element is defined by the ProMillThreadTaperType

Element ID	Data Type	Description
		parameter.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676 .
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_THRM_HOLESET_ARR	Array	Specifies an array of thread holesets. It gives specification of threads to machine. For more information, refer to the section Manufacturing Thread Milling Holeset on page 1513.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion types for this element are:

Element ID	Data Type	Description
		• PRO_TM_TYPE_THREAD_MILLING. For more information, refer to the section Tool Motion — Thread Milling on page 1780. • PRO_TM_TYPE_FOLLOW_CUT. For more information, refer to the section Tool Motion — Follow Cut on page 1770. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion —

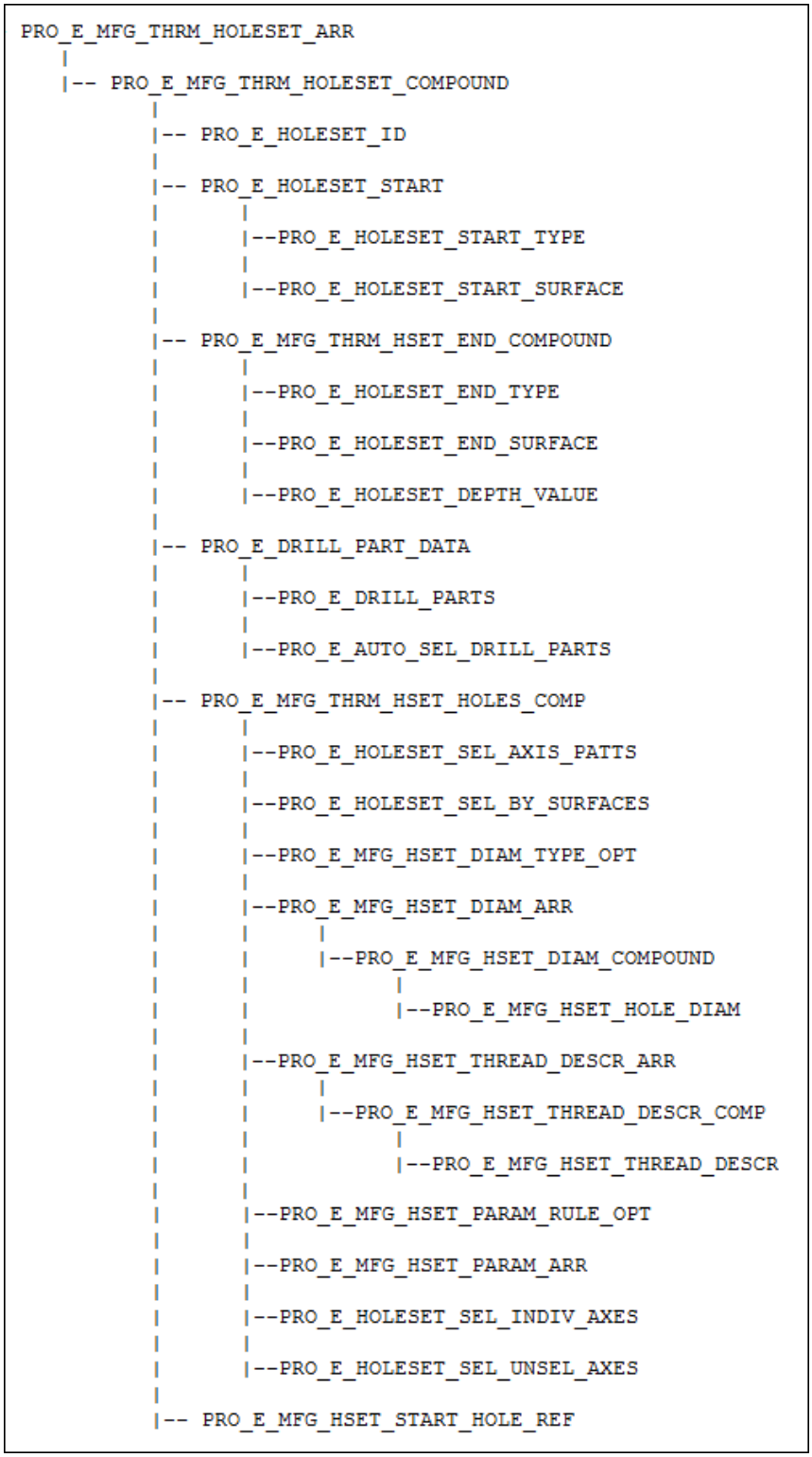
Element ID	Data Type	Description
		Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_RAMP_APPROACH. For more information, refer to the section Tool Motion — Ramp Approach on page 1758. • PRO_TM_TYPE_RAMP_EXIT. For more information, refer to the section Tool Motion — Ramp Exit on page 1760. • PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767. For more information, refer to the section Tool Motion — Auto Cut on page 1766.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to

Element ID	Data Type	Description
		be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Manufacturing Thread Milling Holeset

The element `PRO_E_MFG_THRM_HOLESET_ARR` is documented in the header file `ProMfgElemThreadHoleset.h`, and is shown in the following figure.

Element tree for Manufacturing Thread Milling Holeset



The following table lists the contents of PRO_E_MFG_THRM_HOLESET_ARR element.

Element ID	Data Type	Description
PRO_E_MFG_THRM_HOLESET_ARR	Array	This element specifies an array of thread holesets. It gives specifications about machining references.
PRO_E_MFG_THRM_HOLESET_COMPOUND	Compound	Mandatory element. This compound element specifies the thread holeset definition. **Note** Specify this element only when the holeset array has at least one member
PRO_E_HOLESET_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the holeset type. The valid value for this element is PRO_HOLESET_DRILL_AXES.
PRO_E_HOLESET_START	Compound	Mandatory element. Specifies the holemaking start compound specification.
PRO_E_HOLESET_START_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the start surface option using the enumerated value ProDrillStartType.
PRO_E_HOLESET_START_SURFACE	PRO_VALUE_TYPE_SELECTION	Specifies the starting surface or quilt selection. **Note** This element is mandatory if the element PRO_E_HOLESET_START_TYPE is set to PRO_DRILL_FROM_SURFACE.
PRO_E_MFG_THRM_HSET_END_COMPOUND	Compound	Mandatory element. Specifies the thread depth compound specification.
PRO_E_HOLESET_END_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the end type option. The valid values for this element are: • PRO_DRILL_UPTO_SURFACE • PRO_DRILL_AUTO_END • PRO_DRILL_OFFSET_FROM_START
PRO_E_HOLESET_END_SURFACE	PRO_VALUE_TYPE_SELECTION	Specifies the end surface or quilt

Element ID	Data Type	Description
		selection. **Note** This element is mandatory if the element PRO_E_HOLESET_END_TYPE is set to PRO_DRILL_UPTO_SURFACE.
PRO_E_HOLESET_DEPTH_VALUE	PRO_VALUE_TYPE_DOUBLE	Specifies the depth to the cut thread from the start. **Note** This element is mandatory if the element PRO_E_HOLESET_END_TYPE is set to PRO_DRILL_OFFSET_FROM_START.
PRO_E_DRILL_PART_DATA	Compound	This element gives compound information about components used in depth computation. **Note** Specify this element only if the start or end of machining has to be computed and the following conditions hold true: • The element PRO_E_HOLESET_START_TYPE is set to PRO_DRILL_AUTO_START. • The element PRO_E_HOLESET_END_TYPE is set to PRO_DRILL_AUTO or PRO_DRILL_THRU_ALL • The element PRO_E_HOLESET_DEPTH_TYPE is set to PRO_DRILL_AUTO or PRO_DRILL_THRU_ALL
PRO_E_AUTO_SEL_DRILL_PARTS	PRO_VALUE_TYPE_INT	Mandatory element. This element

Element ID	Data Type	Description
		defines the way in which components are collected. The valid values for this element are: • TRUE— All components of type reference part or workpiece are considered for depth calculation. • FALSE—Only selected components are considered in depth calculation.
PRO_E_DRILL_PARTS	PRO_VALUE_TYPE_SELECTION	Specifies the components selections. This element supports multiple selections. **Note** • This element is mandatory if the element PRO_E_AUTO_SEL_DRILL_PARTS is set to FALSE. • This element is ignored if the element PRO_E_AUTO_SEL_DRILL_PARTS is set to FALSE.
PRO_E_MFG_THRM_HSET_HOLES_COMP	Compound	Mandatory element. This compound element gives the compound information about

Element ID	Data Type	Description
		location of holes. **Note** Specify this element only when at least one of the following have been defined: • PRO_E_HOLESET_SEL_INDIV_AXES • PRO_E_HOLESET_SEL_AXIS_PATTS • PRO_E_HOLESET_SEL_BY_SURFACES • PRO_E_MFG_HSET_DIAM_TYPE_OPT • PRO_E_MFG_HSET_THREAD_DESCR_ARR • PRO_E_MFG_HSET_DIAM_ARR • PRO_E_MFG_HSET_PARAM_ARR
PRO_E_HOLESET_SEL_AXIS_PATTS	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of axes of patterned holes. This element supports multiple selections. **Note** If a pattern leader is selected, all holes in pattern will be collected.
PRO_E_HOLESET_SEL_BY_SURFACES	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of surfaces or quilts with holes. This element supports multiple selections.
PRO_E_MFG_HSET_DIAM_TYPE_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of hole diameters that can be collected in the element PRO_E_MFG_HSET_DIAM_ARR. The type of hole diameter is specified using the enumerated data type ProHolesetDiamType. The valid values are:

Element ID	Data Type	Description
		• PRO_HSET_ALL_DIAMS—This is the default value. Specifies that diameters of both solid surfaces and cosmetic threads can be collected. **Note** If the element PRO_E_MFG_HSET_DIAM_TYPE_OPT is not defined, then by default, the hole diameter of type PRO_HSET_ALL_DIAMS is used. • PRO_HSET_HOLE_DIAMS—Specifies that diameters only of solid surfaces can be collected. • PRO_HSET_THREAD_DIAMS—Specifies that diameters only of cosmetic threads can be collected.
PRO_E_MFG_HSET_DIAM_ARR	Array	Optional element. Specifies an array of diameters of holes to machine.
PRO_E_MFG_HSET_DIAM_COMPOUND	Compound	Optional element. Specifies the compound definition of a hole diameter.
PRO_E_MFG_HSET_HOLE_DIAM	PRO_VALUE_TYPE_DOUBLE	Specifies the diameter of a hole to machine. **Note** This element is a mandatory child element of the element PRO_E_MFG_HSET_DIAM_COMPOUND.
PRO_E_MFG_HSET_THREAD_DESCR_ARR	Array	Optional element. This array element gives thread descriptions of holes to machine.
PRO_E_MFG_HSET_THREAD_DESCR_COMP	Compound	Optional element. Specifies compound definition of a thread description.

Element ID	Data Type	Description
PRO_E_MFG_HSET_THREAD_DESCR	PRO_VALUE_TYPE_WSTRING	Specifies the thread size string.  **Note** This element is a mandatory child of PRO_E_MFG_HSET_THREAD_DESCR_COMP element.
PRO_E_MFG_HSET_PARAM_RULE_OPT	PRO_VALUE_TYPE_INT	Specifies the type of query that must be used to search for holes to machine. The query type is specified using the enumerated data type ProHsetParamRuleOpt. The valid values are: • PRO_HSET_BOOL_OPER_OR—Collects holes that satisfy at least one of the search conditions set for a parameter. PRO_HSET_BOOL_OPER_AND—Collects holes that satisfy all the search conditions set for a parameter. The search conditions and parameters are defined in the elements PRO_E_MFG_HSET_PARAM*.
PRO_E_MFG_HSET_PARAM_ARR	Array	Optional element. Specifies an array of search conditions to collect holes for machining.
PRO_E_MFG_HSET_PARAM_COMPOUND	Compound	Optional element. Specifies a compound element that defines a search condition to match with the user defined parameters in hole features. Each condition defines an expression with user defined parameter name on the left side of the expression and value to compare on the right side.
PRO_E_MFG_HSET_PARAM_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the user

Element ID	Data Type	Description
		defined parameter. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element.
PRO_E_MFG_HSET_PARAM_DTYPE	PRO_VALUE_TYPE_INT	Specifies the data type of the values using the enumerated value ProParamvalueType. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element.
PRO_E_MFG_HSET_PARAM_OPER	PRO_VALUE_TYPE_INT	Specifies the type of expression operator using the enumerated value ProDrillParamOper. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element.
PRO_E_MFG_HSET_PARAM_VAL_DBL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the double data type. **Note** • This element is mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for double data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_DOUBLE). It is ignored for other data types.
PRO_E_MFG_HSET_PARAM_VAL_INT	PRO_VALUE_TYPE_INT	Specifies the value of the integer

Element ID	Data Type	Description
		data type. **Note** • This element is mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for integer data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_INTEGER). It is ignored for other data types.
PRO_E_MFG_HSET_PARAM_VAL_STR	PRO_VALUE_TYPE_WSTRING	Specifies the value of the string data type. **Note** • This element is mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for string data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_STRING). It is ignored for other data types.
PRO_E_MFG_HSET_PARAM_VAL_BOOL	PRO_VALUE_TYPE_INT	Specifies the value of the string data type. **Note** • This element is mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for boolean data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_BOOLEAN). It is ignored for other data types.
PRO_E_HOLESET_SEL_INDIV_AXES	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum axes. This element supports multiple selections.

Element ID	Data Type	Description
		Note This element is mandatory if you have not defined the following elements: • PRO_E_HOLESET_SEL_AXIS_PATTS • PRO_E_HOLESET_SEL_BY_SURFACES • PRO_E_MFG_HSET_THREAD_DESCR_ARR • PRO_E_MFG_HSET_DIAM_ARR • PRO_E_MFG_HSET_PARAM_ARR
PRO_E_HOLESET_SEL_UNSEL_AXES	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of axes of holes to be excluded for machining. This element supports multiple selections.
PRO_E_MFG_HSET_START_HOLE_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the axis selection for the hole to be machined first.

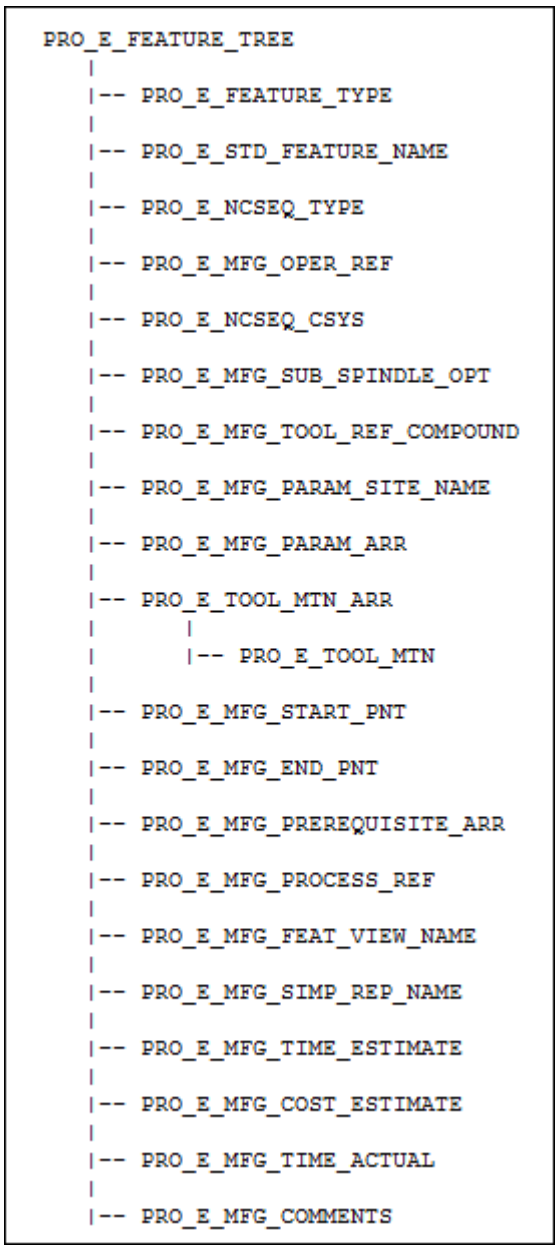
Element Trees: Turning Step

This section describes how to construct and access the element tree for a turning step. It also describes how to create, redefine, and access the properties of these features. Refer to the Creo NC Help for more information on the Turn type Creo NC sequences.

The Turning Element Tree:

The element tree for the turning step is documented in the header file `ProMfgFeatTurning.h`, and is as shown in the following figure:

Element Tree for Turning feature



The following table describes the elements in the element tree for the area turning feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_TURN.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default values for this element

Element ID	Data Type	Description
		are: • Area_Turning_2—For area turning Creo NC sequences. • Profile_Turning_1—For profile turning Creo NC sequences. • Groove_Turning_1—For groove turning Creo NC sequences.
PRO_E_SEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid values for this element are: • PRO_SEQ_AREA_TURN—For an area turning sequence. • PRO_SEQ_GROOVE_TURN—For a groove turning sequence. • PRO_SEQ_PROF_TURN—For a profile turning sequence.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_SEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the Creo NC sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters.

Element ID	Data Type	Description
		Note The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion types for this element are: • PRO_TM_TYPE_AREA_TURNING. For more information, refer to the section Tool Motion — Area and Groove Turning on page 1731. • PRO_TM_TYPE_GROOVE_TURNING. For more information, refer to the section Tool Motion — Area and Groove Turning on page 1731. • PRO_TM_TYPE_PROF_TURNING. For more information, refer to the section Tool Motion — Profile Turning on page 1737. • PRO_TM_TYPE_FOLLOW_CUT. For more information, refer to the section Tool Motion — Follow Cut on page 1770 PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696 • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700.

Element ID	Data Type	Description
		• PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713 • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_RAMP_APPROACH. For more information, refer to the section Tool Motion — Ramp Approach on page 1758. • PRO_TM_TYPE_RAMP_EXIT. For more information, refer to the section Tool Motion — Ramp Exit on page 1760. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762 • PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767. For more information, refer to the section Tool Motion — Auto Cut on page 1766.

Element ID	Data Type	Description
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ ARR	Array	Optional element. This array specifies the Ids of the prerequisiteCreo NC sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_ NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining

Element ID	Data Type	Description
		step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

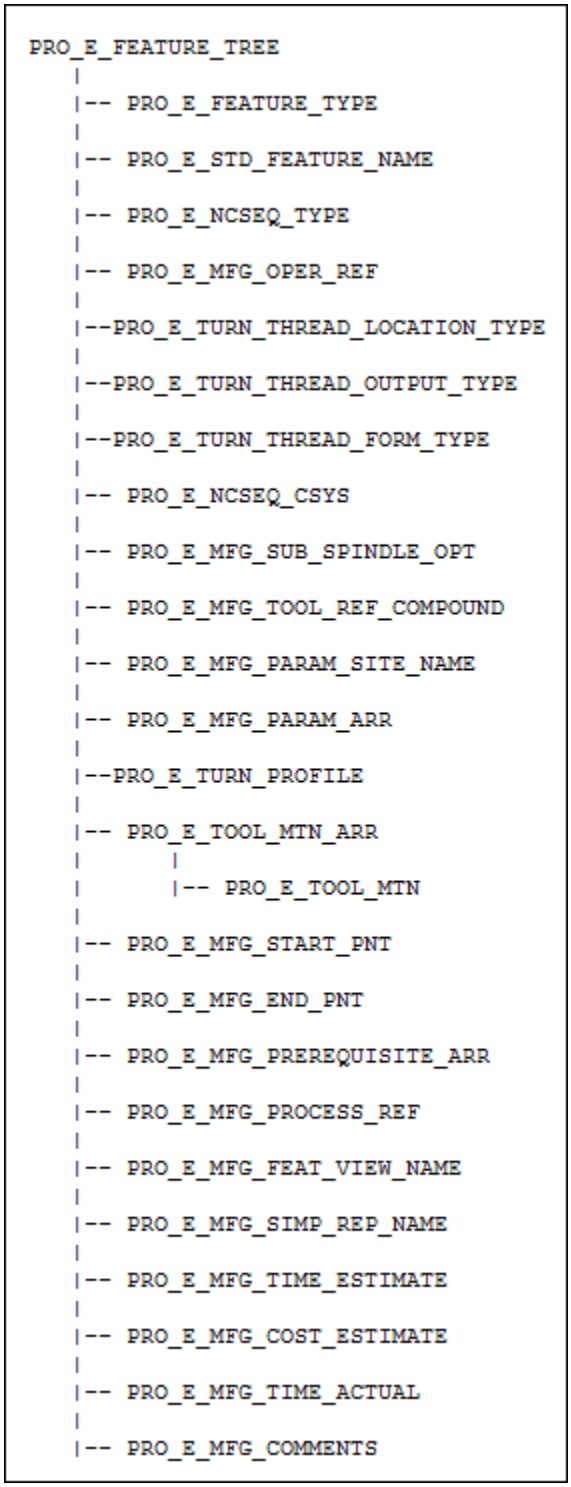
Element Trees: Thread Turning Step

This section describes how to construct and access the element tree for a thread turning step. It also describes how to create, redefine, and access the properties of these features.

The Thread Turning Element Tree:

The element tree for the thread turning Creo NCsequence is documented in the header file `ProMfgFeatTurnThread.h`, and is as shown in the following figure:

Element Tree for Thread Turning feature



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_TURN.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is Thread_Turning_1.
PRO_E_SEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_SEQ_THREAD_TURN
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_TURN_THREAD_LOCATION_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the location of the thread. The valid values for this element are: • PRO_E_TURN_OPTION_OUT—For external threads • PRO_E_TURN_OPTION_IN—For internal threads
PRO_E_TURN_THREAD_OUTPUT_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of output You can specify one of the following valid values for this element: • PRO_E_TURN_THREAD_ISO • PRO_E_TURN_THREAD_AI_MACRO
PRO_E_TURN_THREAD_FORM_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the thread. . The valid values for this element are: • PRO_E_TURN_THREAD_UNIFIED—For Unified threads • PRO_E_TURN_THREAD_GENERAL—For General threads • PRO_E_TURN_THREAD_BUTTRESS—For Buttress threads • PRO_E_TURN_THREAD_ACME—For Acme threads
PRO_E_SEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in

Element ID	Data Type	Description
		different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type `ProSubSpindleOpt`. For more information on the values of `ProSubSpindleOpt`, refer to the section Spindle Types on page 1690
`PRO_E_MFG_TOOL_REF_COMPOUND`	`Compound`	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673 .
`PRO_E_MFG_PARAM_SITE_NAME`	`PRO_VALUE_TYPE_WSTRING`	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
`PRO_E_MFG_PARAM_ARR`	`Array`	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
`PRO_E_TURN_PROFILE`	`PRO_VALUE_TYPE_SELECTION`	Mandatory element. Specifies turn profile selection.
`PRO_E_TOOL_MTN_ARR`	`Array`	Mandatory element. Specifies an array of tool motions.
`PRO_E_TOOL_MTN`	`Compound`	Mandatory compound element. Specifies the tool motion compound specifications. For more information, refer to the section Tool Motion — Auto Cut on page 1766.
`PRO_E_MFG_START_PNT`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
`PRO_E_MFG_END_PNT`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
`PRO_E_MFG_PREREQUISITE_ARR`	`Array`	Optional element. This array specifies the Ids of the prerequisite sequences. For more information,

Element ID	Data Type	Description
		refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.

Element ID	Data Type	Description
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments

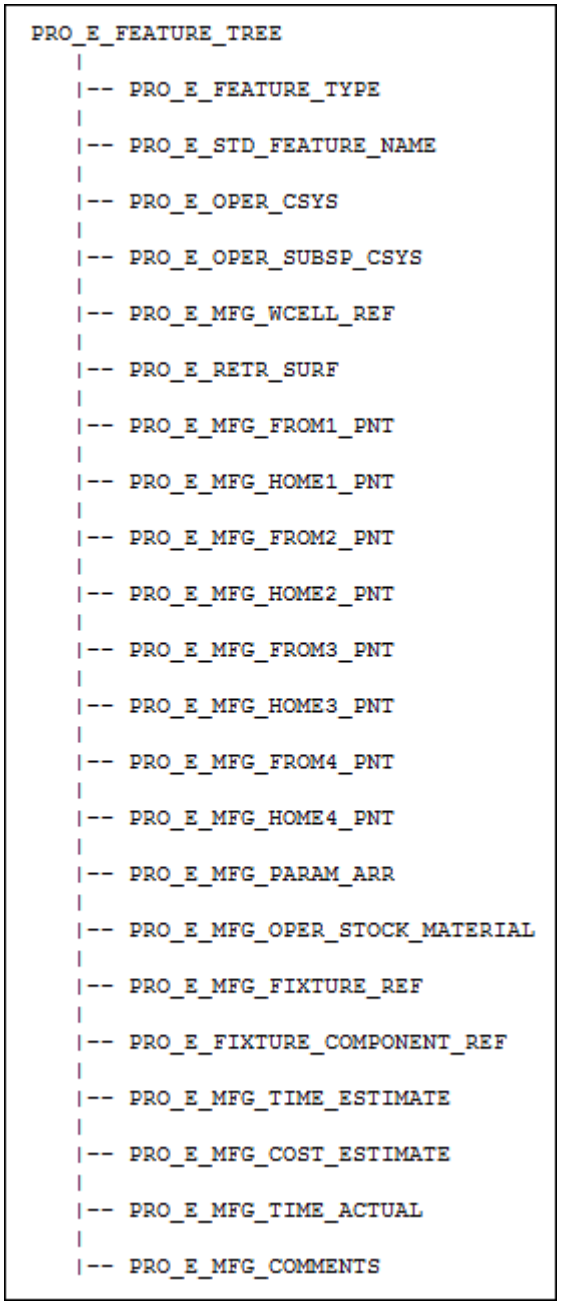
Element Trees: Creo NC Operation Definition

This section describes how to construct and access the element tree for a Creo NC operation definition feature. It also describes how to create, redefine, and access the properties of these features.

The Creo NC Operation Definition Element Tree:

The element tree for the Creo NC operation sequence is documented in the header file `ProMfgFeatOperation.h`, and is as shown in the following figure:

Element Tree for Creo NC operation definition feature



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_OPERATION.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is OP040.

Element ID	Data Type	Description
PRO_E_OPER_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the operation coordinate system for the Creo NC sequence.
PRO_E_OPER_SUBSP_CSYS	PRO_VALUE_TYPE_SELECTION	Optional element. Select a coordinate system geometry item which can be used as a sub-spindle coordinate system.
PRO_E_MFG_WCELL_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the workcell feature selection.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_FROM1_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to start machining by the head 1 tool at the specified position.
PRO_E_MFG_HOME1_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to end machining by the head 1 tool at the specified position.
PRO_E_MFG_FROM2_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to start machining by the head 2 tool at the specified position. **Note** This element is ignored for workcells with single head.
PRO_E_MFG_HOME2_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to end machining by head the 2 tool at the specified position. **Note** This element is ignored for workcells with single head.
PRO_E_MFG_FROM3_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point geometry item selection. Allows to start machining by the head 3 tool at the specified position. **Note** This element is ignored for workcells with number of heads less than 3.

Element ID	Data Type	Description
PRO_E_MFG_HOME3_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point geometry item selection. Allows to end machining by the head 3 tool at the specified position. **Note** This element is ignored for workcells with number of heads less than 3.
PRO_E_MFG_FROM4_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point geometry item selection. Allows to start machining by the head 4 tool at the specified position. **Note** This element is ignored for workcells with number of heads less than 4.
PRO_E_MFG_HOME4_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point geometry item selection. Allows to end machining by the head 4 tool at the specified position. **Note** This element is ignored for workcells with number of heads less than 4.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of applicable manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h.

Element ID	Data Type	Description
		For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree. **Note** For new features, if the parameter array is not specified the default values will be assigned to the corresponding manufacturing parameters of the created feature.
PRO_E_MFG_OPER_STOCK_MATERIAL	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the stock material name.
PRO_E_MFG_FIXTURE_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of a fixture setup feature.
PRO_E_FIXTURE_COMPONENT_REF	Array	Optional element. Specifies an array of operation fixture setup components that can be inserted into the top assembly. This array can be can be specified either in combination with or without a fixture setup reference specified by PRO_E_MFG_FIXTURE_REF. This element supports multiple selections.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.

Element ID	Data Type	Description
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

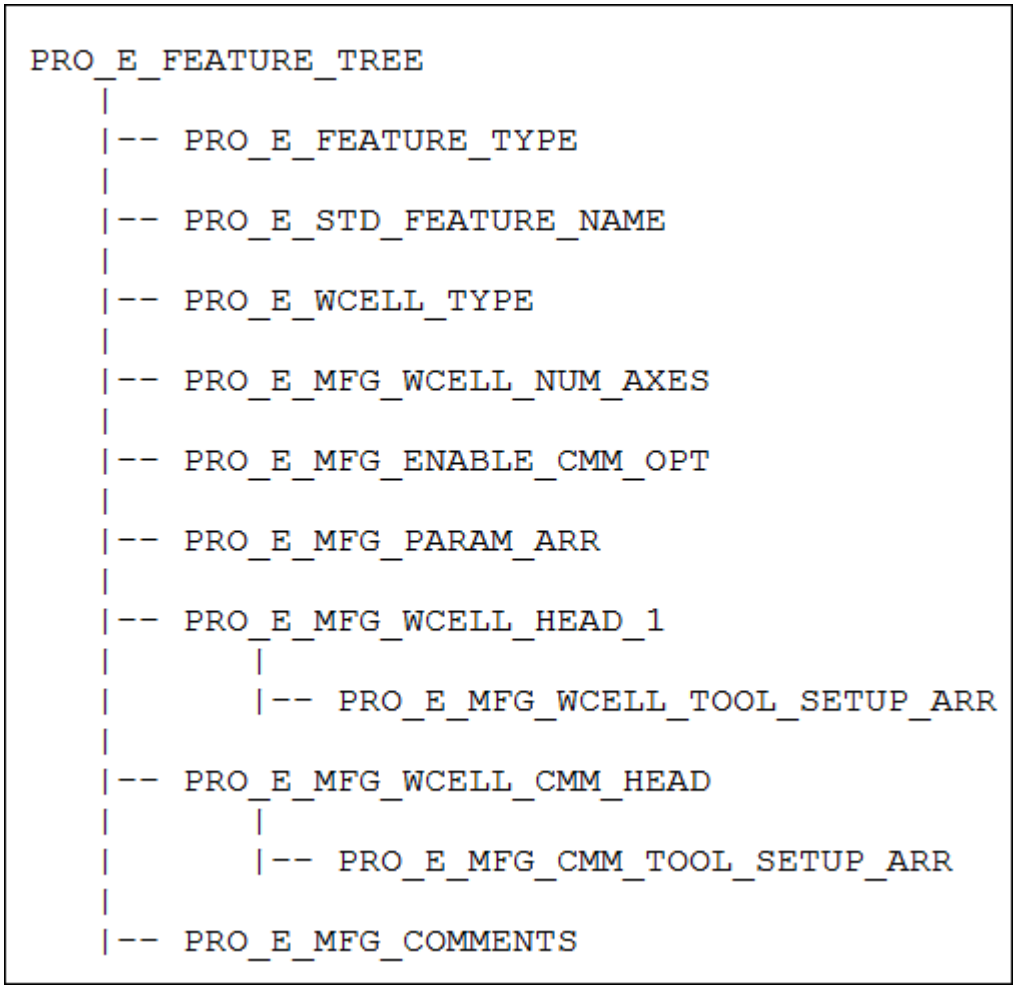
Element Trees: Workcell Definition

This section describes how to construct and access the element tree for a workcell definition. It also describes how to create, redefine, and access the properties of these features.

The Manufacturing WEDM Workcell Element Tree

The element tree for the WEDM workcell type is documented in the header file `ProMfgFeatWcellWedm.h`, and is as shown in the following figure:

Element Tree for WEDM workcell feature



The following table describes the elements in the element tree for the WEDM workcell feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_WORKCELL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is WEDM01.
PRO_E_WCELL_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the workcell used for the Creo NC sequence. The valid value for this element is PRO_WCELL_WEDM.
PRO_E_MFG_WCELL_NUM_AXES	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the number of controlled axes

Element ID	Data Type	Description
		(number of programmable motion directions). The valid values for this element are: • PRO_WCELL_2_AXIS • PRO_WCELL_4_AXIS
PRO_E_MFG_ENABLE_CMM_OPT	PRO_VALUE_TYPE_INT	Allows enabling/disabling of the tool head for the creation of CMM sequences. The valid values for this element are: • PRO_B_TRUE: Enables tool head with CMM probes and allows the creation of CMM sequences • PRO_B_FALSE: Disables CMM tool head and creation of CMM sequences. **Note** This element is optional , when CMM tool head is not defined.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_WCELL_HEAD_1	Compound	Optional element. Specifies the tool head compound definition.
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the tool setup array. For more information, refer to the section Element Trees: Tool Setup on page 1683.
PRO_E_MFG_WCELL_CMM_HEAD	Compound	Optional element. Specifies the CMM probes head compound definition. **Note** This element is ignored if PRO_E_MFG_ENABLE_CMM_OPT is set to PRO_B_FALSE.
PRO_E_MFG_CMM_TOOL_SETUP_ARR	Array	Optional element. Specifies the CMM probes setup array. For more information, refer to the section Element Trees: CMM Probe Setup on page 1686.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the workcell comments.

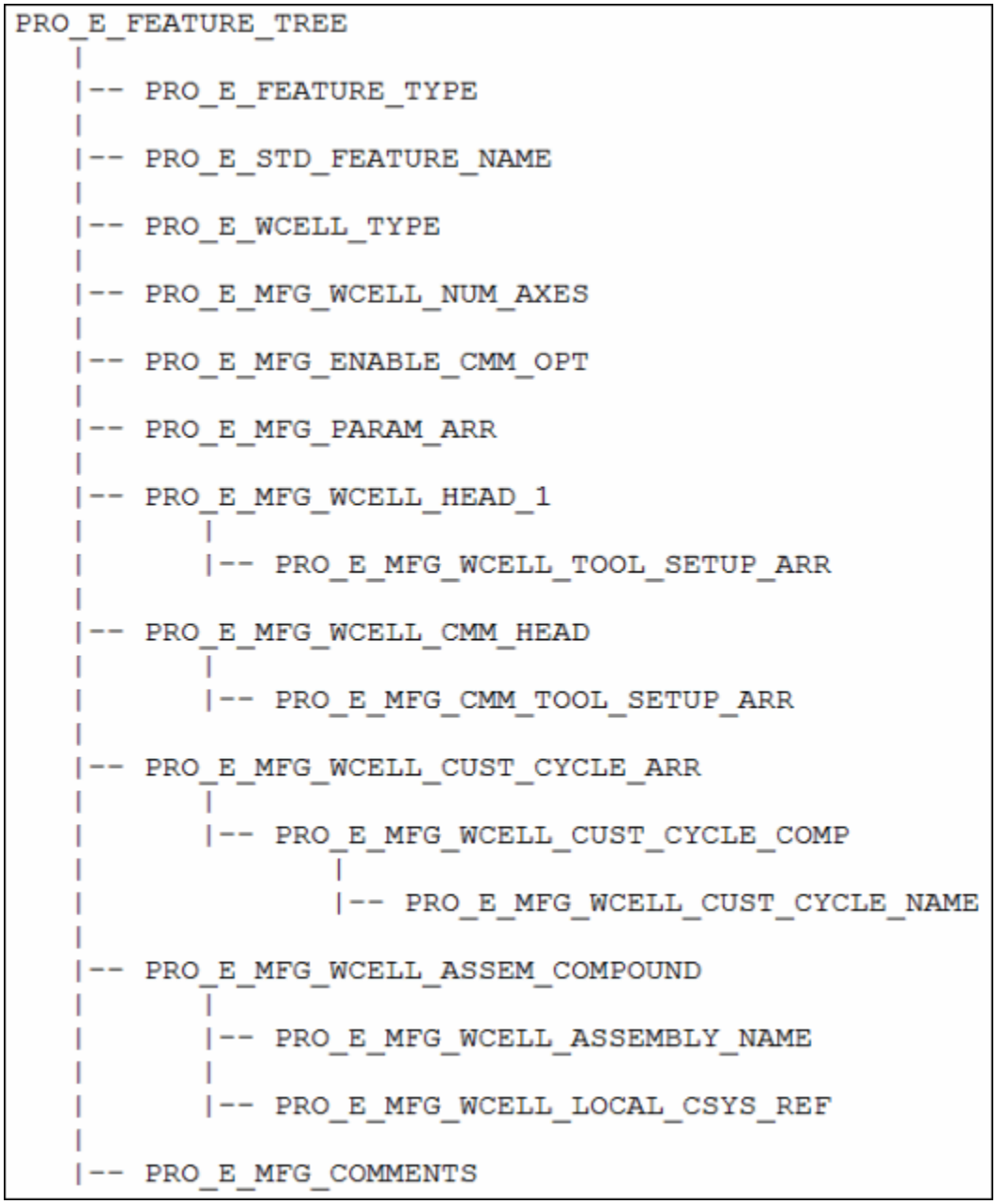
Element Trees: Manufacturing Mill Workcell

This section describes how to construct and access the element tree for manufacturing mill workcell feature. It also describes how to create, redefine, and access the properties of these features.

The Mill Workcell Feature Element Tree:

The element tree for the manufacturing mill workcell feature is documented in the header file `ProMfgFeatWcellMill.h`, and is as shown in the following figure:

Element Tree for Mill Workcell feature



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_WORKCELL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is MILL01.

Element ID	Data Type	Description
PRO_E_WCELL_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the workcell used for the Creo NC sequence. The valid value for this element is PRO_WCELL_MILL.
PRO_E_MFG_WCELL_NUM_AXES	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the number of controlled axes (number of programmable motion directions). The valid values for this element are: • PRO_WCELL_3_AXIS • PRO_WCELL_4_AXIS • PRO_WCELL_5_AXIS
PRO_E_MFG_ENABLE_CMM_OPT	PRO_VALUE_TYPE_INT	Specifies enabling/disabling of the tool head for the creation of CMM sequences. The valid values for this element are: • PRO_B_TRUE: Enables tool head with CMM probes and allows the creation of CMM sequences • PRO_B_FALSE: Disables CMM tool head and creation of the CMM sequences. **Note** This element is optional, when CMM tool head is not defined or is disabled.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677 .
PRO_E_MFG_WCELL_HEAD_1	Compound	Optional element. Specifies the tool head compound definition.
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the tool setup array. For more information, refer to the section Element Trees: Tool Setup on page 1683.
PRO_E_MFG_WCELL_CMM_HEAD	Compound	Optional compound element. Specifies the CMM probes head

Element ID	Data Type	Description
		compound definition. **Note** This element is ignored if PRO_E_MFG_ENABLE_CMM_OPT is set to PRO_B_FALSE.
PRO_E_MFG_CMM_TOOL_SETUP_ARR	Array	Optional element. Specifies the CMM probes setup array. For more information, refer to the section Element Trees: CMM Probe Setup on page 1686 .
PRO_E_MFG_WCELL_CUST_CYCLE_ARR	Array	Optional element. Specifies the array of custom cycle names (to be used by holemaking sequences). For holemaking sequences, For more information, refer to the section Manufacturing Holemaking Step on page 1578.
PRO_E_MFG_WCELL_CUST_CYCLE_COMP	Compound	Optional compound element. Specifies the compound definition of a custom cycle name.
PRO_E_MFG_WCELL_CUST_CYCLE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the custom cycle name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_CUST_CYCLE_COMP element.
PRO_E_MFG_WCELL_ASSEM_COMPOUND	Compound	Optional compound element. Specifies the Simulation assembly compound definition.
PRO_E_MFG_WCELL_ASSEMBLY_NAME	Assembly Name	Specifies the simulation assembly model name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.

Element ID	Data Type	Description
PRO_E_MFG_WCELL_LOCAL_CSYS_REF	PRO_VALUE_TYPE_SELECTION	Specifies the coordinate system datum feature. It will be aligned with the simulation assembly coordinate system during machining simulation. **Note** • This element is a mandatory child element of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the workcell comments.

Example 1: Creating or Redefining a Tool from a File

The sample code in `UgMfgToolFileReadWrite.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_mfg` shows how to create a tool or redefine it from a file and save the information about the tool to a file.

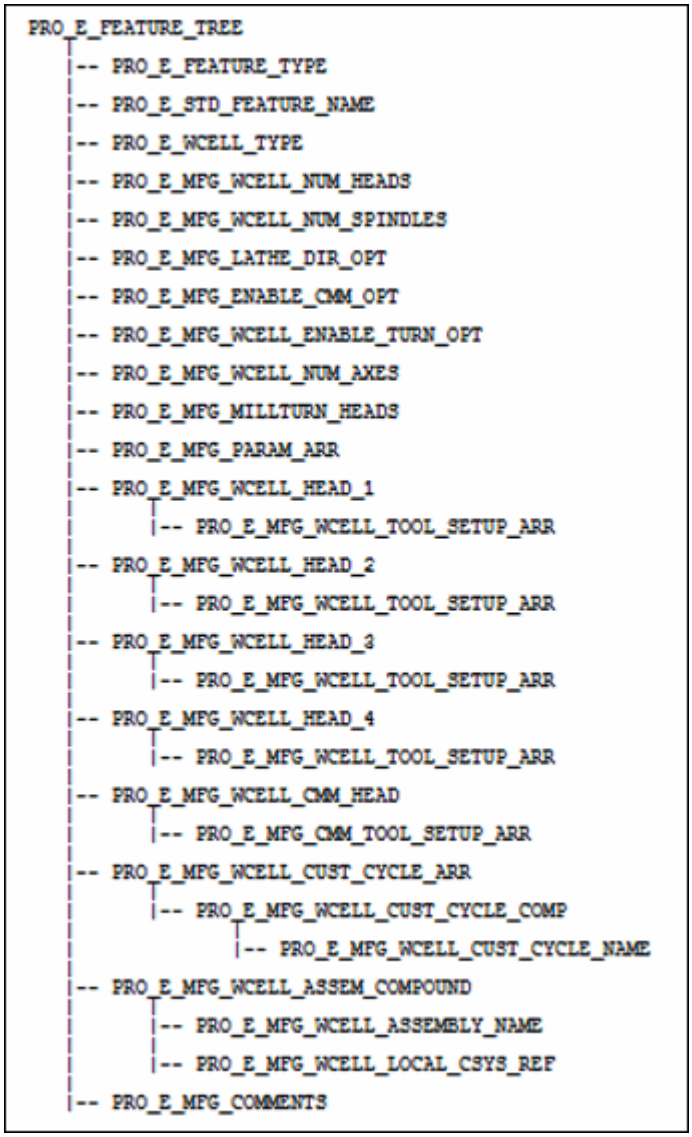
Element Trees: Manufacturing Mill/Turn Workcell

This section describes how to construct and access the element tree for a mill turn workcell. It also describes how to create, redefine, and access the properties of these features.

The Mill/Turn Workcell Element Tree:

The element tree for the mill/turn workcell is documented in the header file `ProMfgFeatWcellMillTurn.h`, and is as shown in the following figure:

Element Tree for Mill/Turn Workcell



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_WORKCELL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is MILL01.
PRO_E_WCELL_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the workcell used for the Creo NC sequence. The valid value for this element is PRO_WCELL_MILL_N_TRN.

Element ID	Data Type	Description
PRO_E_MFG_WCELL_NUM_HEADS	PRO_VALUE_TYPE_INT	Optional element. Specifies the number of tool heads (turrets). The valid values are: • PRO_MFG_ONE_HEAD_WCELL—This is the default value. • PRO_MFG_TWO_HEAD_WCELL
PRO_E_MFG_WCELL_NUM_SPINDLES	PRO_VALUE_TYPE_INT	Optional element. Specifies the number of spindles to be used for the feature creation. The valid values for this element are defined in the enumerated type ProMfgWcellNumSpindles and are as follows: • PRO_MFG_ONE_SPINDLE_WCELL—This is the default value. • PRO_MFG_TWO_SPINDLE_WCELL
PRO_E_MFG_LATHE_DIR_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the lathe orientation . The valid values for this element are: • PRO_WCELL_LATHE_HORIZONTAL—This is the default value. • PRO_WCELL_LATHE_VERTICAL
PRO_E_MFG_WCELL_ENABLE_TURN_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the enabling/disabling of the turning machine. The valid values for this element are: • PRO_B_TRUE—Enables the turning machining on the workcell. This is he default value. • PRO_B_FALSE—Disables the turning machining on the workcell.
PRO_E_MFG_ENABLE_CMM_OPT	PRO_VALUE_TYPE_INT	Optional element. Initializes a CMM operation. The valid values for this element are: • PRO_B_TRUE: Enables the tool head with CMM probes and allows the creation of CMM sequences. • PRO_B_FALSE: Disables the CMM tool head and allows the creation of CMM sequences.
PRO_E_MFG_WCELL_NUM_AXES	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the number of controlled axes enabled

Element ID	Data Type	Description
		for milling sequences. The valid values for this element are: • PRO_WCELL_3_AXIS • PRO_WCELL_4_AXIS • PRO_WCELL_5_AXIS
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_WCELL_HEAD_1	Compound	Optional compound element. Specifies the compound definition for tool head 1.
PRO_E_MFG_WCELL_HEAD_2	Compound	Optional element. Specifies the compound definition for tool head 2. **Note** This element is ignored if you have set the element PRO_E_MFG_WCELL_NUM_HEADS to PRO_MFG_ONE_HEAD_WCELL value.
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the Tool setup array. For more information, refer to the section Element Trees: Tool Setup on page 1683.
PRO_E_MFG_WCELL_HEAD_3	Compound	Optional element. Specifies the tools specification for head 3. **Note** This element is ignored for workcells with number of heads less than 3 that is, when the element PRO_E_MFG_WCELL_NUM_HEADS is set to PRO_MFG_ONE_HEAD_WCELL or PRO_MFG_TWO_HEAD_WCELL.
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the Tool setup array. For more information, refer to the section Element Trees: Tool Setup on page 1683.
PRO_E_MFG_WCELL_HEAD_4	Compound	Optional element. Specifies the

Element ID	Data Type	Description
		tools specification for head 4. **Note** This element is ignored for workcells with number of heads less than 4 that is, when the element PRO_E_MFG_WCELL_NUM_HEADS is set to PRO_MFG_ONE_HEAD_WCELL, PRO_MFG_TWO_HEAD_WCELL or PRO_MFG_THREE_HEAD_WCELL.
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the Tool setup array. For more information, refer to the section Element Trees: Tool Setup on page 1683.
PRO_E_MFG_WCELL_CMM_HEAD	Compound	Optional compound element. Specifies the CMM probes head compound definition. **Note** This element is ignored if PRO_E_MFG_ENABLE_CMM_OPT is set to PRO_B_FALSE.
PRO_E_MFG_CMM_TOOL_SETUP_ARR	Array	Optional element. Specifies the CMM probes setup array. For more information, refer to the section Element Trees: CMM Probe Setup on page 1686 .
PRO_E_MFG_WCELL_CUST_CYCLE_ARR	Array	Optional element. Specifies the array of custom cycle names (to be used by holemaking sequences). For holemaking sequences, For more information, refer to the section Manufacturing Holemaking Step on page 1578.
PRO_E_MFG_WCELL_CUST_CYCLE_COMP	Compound	Optional compound element. Specifies the compound definition of a custom cycle name.

Element ID	Data Type	Description
PRO_E_MFG_WCELL_CUST_CYCLE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the custom cycle name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_CUST_CYCLE_COMP element.
PRO_E_MFG_WCELL_ASSEM_COMPOUND	Compound	Optional compound element. Specifies the simulation assembly compound definition.
PRO_E_MFG_WCELL_ASSEMBLY_NAME	Assembly Name	Specifies the simulation assembly model name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_WCELL_LOCAL_CSYS_REF	PRO_VALUE_TYPE_SELECTION	Specifies the coordinate system datum feature. It will be aligned with the simulation assembly coordinate system during machining simulation. **Note** This element is a mandatory child element of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the workcell comments.

PRO_E_MFG_MILLTURN_HEADS Element

The compound element PRO_E_MFG_MILLTURN_HEADS contains elements that can be used to define the parameters for the turret head. The elements of PRO_E_MFG_MILLTURN_HEADS are as follows:

- PRO_E_MFG_MILLTURN_HEAD_1—Contains the options for the first turret.
- PRO_E_MFG_MILLTURN_HEAD_2—Contains the options for the second turret.
- PRO_E_MFG_MILLTURN_HEAD_3—Contains the options for the third turret.

- PRO_E_MFG_MILLTURN_HEAD_4—Contains the options for the fourth turret.
- PRO_E_MILLTURN_TOOL_ROT_DIR_OPT

Element Tree for PRO_E_MFG_MILLTURN_HEADS element

```
|-- PRO_E_MFG_MILLTURN_HEADS
|        |
|        |--PRO_E_MFG_MILLTURN_HEAD_1
|        |       |
|        |       |--PRO_E_WCELL_HEAD_MILL_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_TURN_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_FLASH_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_ROT_POS_OPT
|        |
|        |--PRO_E_MFG_MILLTURN_HEAD_2
|        |       |
|        |       |--PRO_E_WCELL_HEAD_MILL_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_TURN_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_FLASH_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_ROT_POS_OPT
|        |
|        |--PRO_E_MFG_MILLTURN_HEAD_3
|        |       |
|        |       |--PRO_E_WCELL_HEAD_MILL_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_TURN_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_FLASH_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_ROT_POS_OPT
|        |
|        |--PRO_E_MFG_MILLTURN_HEAD_4
|        |       |
|        |       |--PRO_E_WCELL_HEAD_MILL_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_TURN_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_FLASH_OPT
|        |       |
|        |       |--PRO_E_WCELL_HEAD_ROT_POS_OPT
|        |
|        |--PRO_E_MILLTURN_TOOL_ROT_DIR_OPT
|
```

The elements PRO_E_MFG_MILLTURN_HEAD_1, PRO_E_MFG_MILLTURN_HEAD_2, PRO_E_MFG_MILLTURN_HEAD_3 and PRO_E_MFG_MILLTURN_HEAD_4 contain the following elements:

Element ID	Data Type	Description
PRO_E_WCELL_HEAD_MILL_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the turret milling option. The valid values for this element are: • PRO_B_TRUE—Enables the milling operations for the turret head. • PRO_B_FALSE—Disables the milling operations for the turret head.
PRO_E_WCELL_HEAD_TURN_OPT	PRO_VALUE_TYPE_INT	Specifies the enabling/disabling of the turning option for the turret. The valid values for this element are: • PRO_B_TRUE—Enables the turning operations for the turret. This is the default value. • PRO_B_FALSE—Disables the turning operations for the turret. **Note** This element is ignored if the element PRO_E_MFG_WCELL_ENABLE_TURN_OPT is set to PRO_B_FALSE.
PRO_E_WCELL_HEAD_FLASH_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the turret flash tool option. The valid values for this element are: • PRO_B_TRUE—Enables the flash tool for the turret. • PRO_B_FALSE—Disables the flash tool for the turret.
PRO_E_WCELL_HEAD_ROT_POS_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the turret rotation positioning option. The valid values for this element are: • PRO_B_TRUE—Enables the rotation positioning for the turret. • PRO_B_FALSE—Disables the rotation positioning for the turret.

Element Trees: Manufacturing Lathe Workcell

This section describes how to construct and access the element tree for a lathe workcell. It also describes how to create, redefine, and access the properties of these features.

The Lathe Workcell Element Tree:

The element tree for the lathe workcell is documented in the header file `ProMfgFeatWcellLathe.h`, and is as shown in the following figure:

Element Tree for Lathe Workcell

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_WCELL_TYPE
    |
    |-- PRO_E_MFG_WCELL_NUM_HEADS
    |
    |-- PRO_E_MFG_WCELL_NUM_SPINDLES
    |
    |-- PRO_E_MFG_LATHE_DIR_OPT
    |
    |-- PRO_E_MFG_ENABLE_CMM_OPT
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_MFG_WCELL_HEAD_1
    |       |
    |       |-- PRO_E_MFG_WCELL_TOOL_SETUP_ARR
    |
    |-- PRO_E_MFG_WCELL_HEAD_2
    |       |
    |       |-- PRO_E_MFG_WCELL_TOOL_SETUP_ARR
    |
    |-- PRO_E_MFG_WCELL_CMM_HEAD
    |       |
    |       |-- PRO_E_MFG_CMM_TOOL_SETUP_ARR
    |
    |-- PRO_E_MFG_WCELL_CUST_CYCLE_ARR
    |       |
    |       |-- PRO_E_MFG_WCELL_CUST_CYCLE_COMP
    |               |
    |               |-- PRO_E_MFG_WCELL_CUST_CYCLE_NAME
    |
    |-- PRO_E_MFG_WCELL_ASSEM_COMPOUND
    |       |
    |       |-- PRO_E_MFG_WCELL_ASSEMBLY_NAME
    |       |
    |       |-- PRO_E_MFG_WCELL_LOCAL_CSYS_REF
    |
    |-- PRO_E_MFG_COMMENTS
```

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_WORKCELL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is LATHE01.
PRO_E_WCELL_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the

Element ID	Data Type	Description
		type of the workcell used for the Creo NC sequence. The valid value for this element is PRO_WCELL_LATHE.
PRO_E_MFG_WCELL_NUM_HEADS	PRO_VALUE_TYPE_INT	Optional element. Specifies the number of tool heads (turrets). The valid values are: • PRO_MFG_ONE_HEAD_WCELL—This is the default value. • PRO_MFG_TWO_HEAD_WCELL
PRO_E_MFG_WCELL_NUM_SPINDLES	PRO_VALUE_TYPE_INT	Optional element. Specifies the number of spindles to be used for the feature creation. The valid values for this element are defined in the enumerated type ProMfgWcellNumSpindles and are as follows: • PRO_MFG_ONE_SPINDLE_WCELL—This is the default value. • PRO_MFG_TWO_SPINDLE_WCELL
PRO_E_MFG_LATHE_DIR_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the lathe orientation . The valid values for this element are: • PRO_WCELL_LATHE_HORIZONTAL—This is the default value. • PRO_WCELL_LATHE_VERTICAL
PRO_E_MFG_ENABLE_CMM_OPT	PRO_VALUE_TYPE_INT	Optional element. Initializes a CMM operation. The valid values for this element are: • PRO_B_TRUE: Enables the tool head with CMM probes and allows the creation of CMM sequences. • PRO_B_FALSE: Disables the CMM tool head and allows the creation of CMM sequences.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_WCELL_HEAD_1	Compound	Optional compound element. Specifies the compound definition for tool head 1.
PRO_E_MFG_WCELL_HEAD_2	Compound	Optional element. Specifies the

Element ID	Data Type	Description
		compound definition for tool head 2. **Note** This element is ignored if you have set the element PRO_E_MFG_WCELL_NUM_HEADS to PRO_MFG_ONE_HEAD_WCELL value.
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the Tool setup array. For more information, refer to the section Element Trees: Tool Setup on page 1683.
PRO_E_MFG_WCELL_CMM_HEAD	Compound	Optional compound element. Specifies the CMM probes head compound definition. **Note** This element is ignored if PRO_E_MFG_ENABLE_CMM_OPT is set to PRO_B_FALSE.
PRO_E_MFG_CMM_TOOL_SETUP_ARR	Array	Optional element. Specifies the CMM probes setup array. For more information, refer to the section Element Trees: CMM Probe Setup on page 1686.
PRO_E_MFG_WCELL_CUST_CYCLE_ARR	Array	Optional element. Specifies the array of custom cycle names (to be used by holemaking sequences). For holemaking sequences, For more information, refer to the section Manufacturing Holemaking Step on page 1578
PRO_E_MFG_WCELL_CUST_CYCLE_COMP	Compound	Optional compound element. Specifies the compound definition of a custom cycle name.
PRO_E_MFG_WCELL_CUST_CYCLE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the custom cycle name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_CUST_CYCLE_COMP element.
PRO_E_MFG_WCELL_ASSEM_COMPOUND	Compound	Optional compound element. Specifies the Simulation assembly

Element ID	Data Type	Description
		compound definition.
PRO_E_MFG_WCELL_ASSEMBLY_NAME	Assembly Name	Specifies the simulation assembly model name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_WCELL_LOCAL_CSYS_REF	PRO_VALUE_TYPE_SELECTION	Specifies the coordinate system datum feature. It will be aligned with the simulation assembly coordinate system during machining simulation. **Note** This element is a mandatory child element of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the workcell comments.

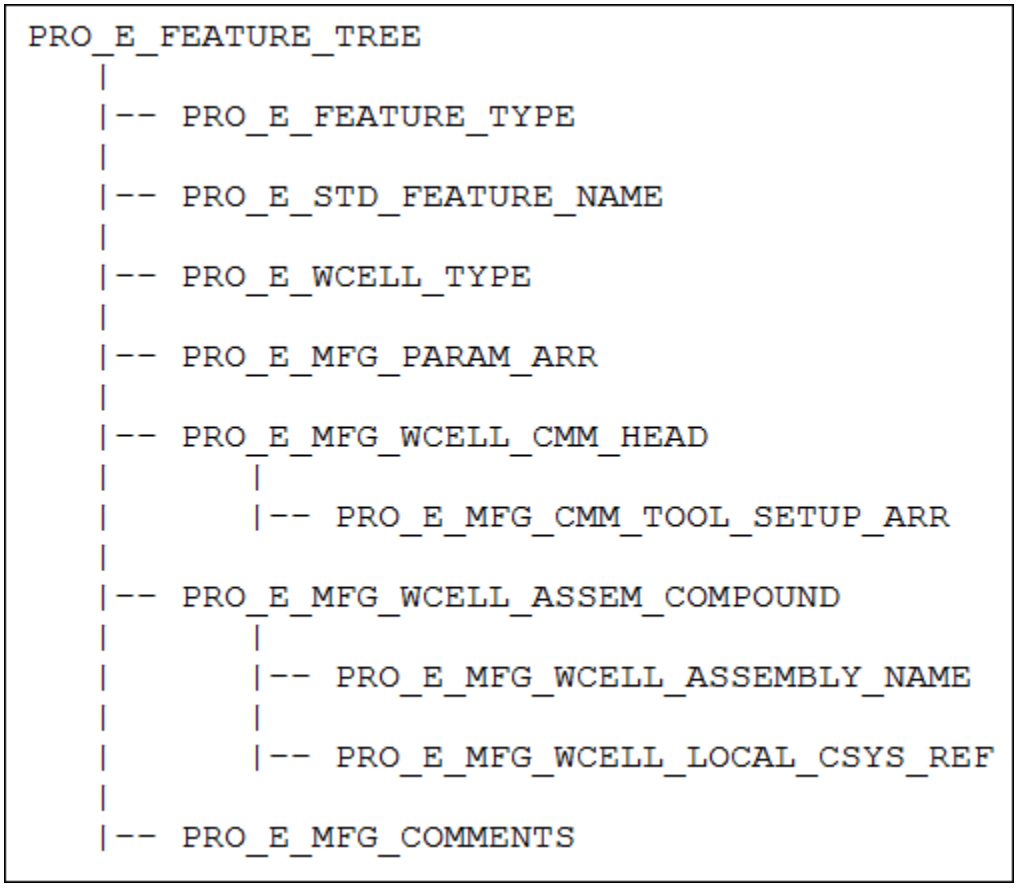
Element Trees: Manufacturing CMM Workcell

This section describes how to construct and access the element tree for a CMM workcell. It also describes how to create, redefine, and access the properties of these features.

The CMM Workcell Element Tree:

The element tree for the CMM workcell is documented in the header file ProMfgFeatWcellCmm.h, and is as shown in the following figure:

Element Tree for CMM Workcell



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_WORKCELL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value for this element is Add default value.
PRO_E_WCELL_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the workcell used for the Creo NC sequence. The valid value for this element is PRO_WCELL_CMM.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677 .
PRO_E_MFG_WCELL_CMM_HEAD	Compound	Optional compound element. Specifies the CMM probes head compound definition.

Element ID	Data Type	Description
PRO_E_MFG_CMM_TOOL_SETUP_ARR	Array	Optional element. Specifies the CMM probes setup array. For more information, refer to the section The CMM probe Setup Element Tree: on page 1686.
PRO_E_MFG_WCELL_ASSEM_COMPOUND	Compound	Optional compound element. Specifies the simulation assembly compound definition.
PRO_E_MFG_WCELL_ASSEMBLY_NAME	Assembly Name	Specifies the simulation assembly model name. **Note** This element is a mandatory child of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_WCELL_LOCAL_CSYS_REF	PRO_VALUE_TYPE_SELECTION	Specifies the coordinate system datum feature. It will be aligned with the simulation assembly coordinate system during machining simulation. **Note** This element is a mandatory child element of PRO_E_MFG_WCELL_ASSEM_COMPOUND element.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the workcell comments.

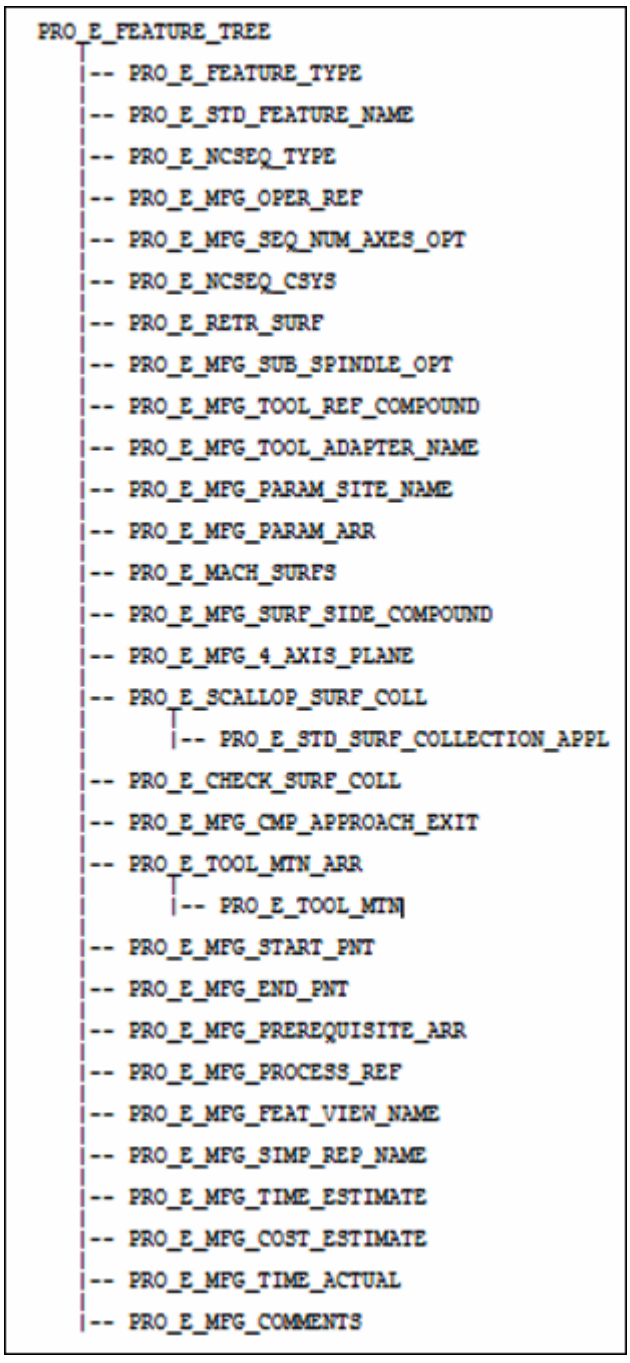
Element Trees: Profile Milling Step

This section describes how to construct and access the element tree for a profile milling step. It also describes how to create, redefine, and access the properties of these features.

The Profile Milling Element Tree:

The element tree for the profile milling Creo NC sequence is documented in the header file ProMfgFeatProfMilling.h, and is as shown in the following figure:

Element Tree for Profile Milling Sequence



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence.

Element ID	Data Type	Description
		The default value is Profile_Milling_1.
PRO_E_SEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_SEQ_PROF_SURF_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_MFG_SEQ_NUM_AXES_OPT	PRO_VALUE_TYPE_INT	Specifies the number of controlled axes. The valid values are: • 3—Applicable if work center allows 3-axis machining. This is the default value. • 4—Applicable if work center allows 4-axis or 5-axis machining. It is used for machining with the tool axis parallel to the plane specified in PRO_E_MFG_4_AXIS_PLANE. • 5—Applicable if work center allows 5-axis machining.
PRO_E_SEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673 .
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the

Element ID	Data Type	Description
NAME		tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677 .
PRO_E_MACH_SURFS	Compound	Mandatory element. Specifies the machining surfaces compound definition.
PRO_E_MFG_SURF_SIDE_COMPOUND	Compound	Optional compound element. Specifies the surfaces side compound definition. For more information, refer to the section Manufacturing Surface Side on page 1566.
PRO_E_MFG_4_AXIS_PLANE	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum plane or planar surface. **Note** • This element is mandatory for 4-axis machining (PRO_E_MFG_SEQ_NUM_AXES_OPT set to 4). • This element is ignored for 3-axis and 5-axis machining.
PRO_E_SCALLOP_SURF_COLL	Compound	Optional compound element. Specifies the scallop surfaces compound definition.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Optional element. Specifies the scallop surfaces collection.
PRO_E_CHECK_SURF_COLL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the check surfaces compound definition. For more information, refer to the section Checking Surfaces on page 1687.
PRO_E_MFG_CMP_APPROACH_EXIT	Compound	Optional element. Specifies the approach and exit compound definition. For more information, refer to the section Approach and Exit on page 1689.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Optional element. Specifies the tool motion compound specification and the applicable tool motion types for this element are: • Tool Motion — Profile Mill Cut on page 1764 • Tool Motion — Follow Cut on page 1770 • Tool Motion — Follow Curve on page 1694 • Tool Motion — Go To Point on page 1696 • Tool Motion — Go Delta on page 1700 • Tool Motion — Go Home on page 1704 • Tool Motion — Plunge on page 1772 • Tool Motion — Go Retract on page 1708 • Tool Motion — Tangent Approach on page 1726 • Tool Motion — Tangent Exit on page 1728 • Tool Motion — Normal Approach on page 1710 • Tool Motion — Normal Exit on page 1713 • Tool Motion — Lead In on page 1706 • Tool Motion — Lead Out on page 1715 • Tool Motion — Helical Approach on page 1717 • Tool Motion — Helical Exit on page 1720 • Tool Motion — Follow Curve on page 1694—For 4 and 5 axis machining. • Approach Along Tool Axis on page 1690 • Exit Along Tool Axis on page 1692—For 4 and 5 axis

Element ID	Data Type	Description
		machining. **Note** The Follow Cut motion must be placed just after the element PRO_TM_TYPE_PROFILE_MILL_CUT, motion or another Follow Cut motion.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.

Element ID	Data Type	Description
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Manufacturing Surface Side

The element PRO_E_MFG_SURF_SIDE_COMPOUND is documented in the header file ProMfgElemSurfSide.h, and is as shown in the following figure:

```
-- PRO_E_MFG_SURF_SIDE_COMPOUND
             |
             |-- PRO_E_MFG_SURF_SIDE_TOLERANCE
             |
             |-- PRO_E_MFG_SURF_SIDE_FLIP_QUILTS
```

The following table describes the elements in the element tree for the surface side feature.

PRO_E_MFG_SURF_SIDE_COMPOUND	Compound	Specifies the surface Side compound.
PRO_E_MFG_SURF_SIDE_TOLERAE	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the tolerance used for composing groups of adjacent quilts. **Note** The value for this element should be nonnegative.
PRO_E_MFG_SURF_SIDE_FLIP_QUILTS	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of quilts to be flipped. This element supports multiple selections.

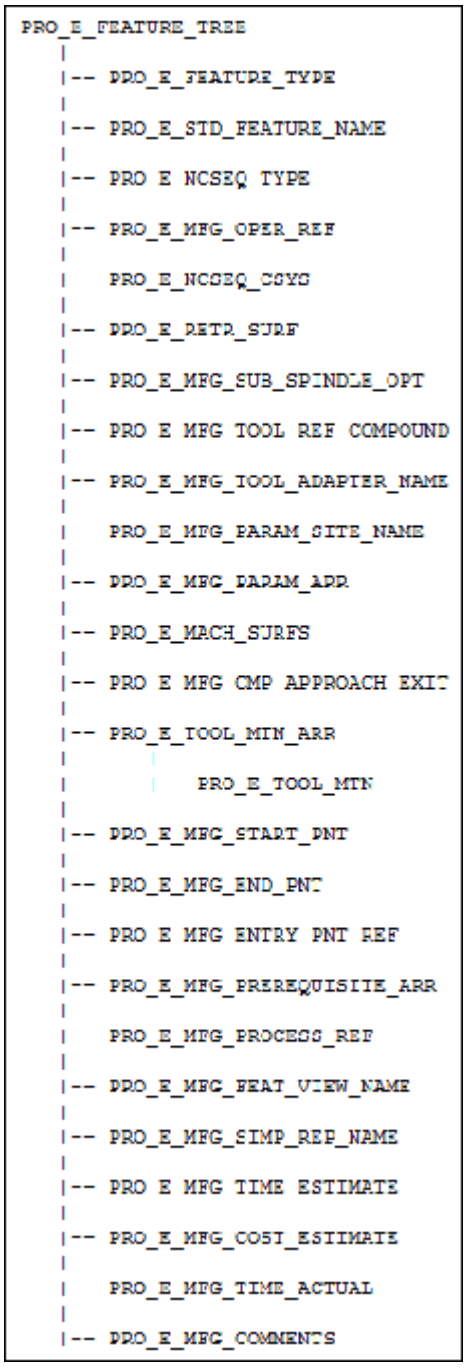
Element Trees: Face Milling Step

This section describes how to construct and access the element tree for a Face Milling feature. It also describes how to create, redefine, and access the properties of these features.

The Face Milling Element Tree:

The element tree for the face milling feature is documented in the header file `ProMfgFeatFacing.h`, and is as shown in the following figure:

Element Tree for Face Milling feature



The following table lists the contents of `PRO_E_FEATURE_TREE` element.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Face_Milling_1.
PRO_E_SEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_SEQ_FACE_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_SEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676 .
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters.

Element ID	Data Type	Description
		Note The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MACH_SURFS	Compound	Mandatory compound element. Specifies the machining surfaces compound definition. For more information, refer to the section Element Trees: Machining Surfaces on page 1573.
PRO_E_MFG_CMP_APPROACH_EXIT	Compound	Optional compound element. Specifies the approach and exit compound definition. For more information, refer to the section Approach and Exit on page 1689.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion types for this element are : • PRO_TM_TYPE_FACE_MILLING. For more information, refer to the section Tool Motion — Face Milling on page 1777. • PRO_TM_TYPE_FOLLOW_CUT. For more information, refer to the section Tool Motion — Follow Cut on page 1770. • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to

Element ID	Data Type	Description
		the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_HELICAL_APPROACH. For more

Element ID	Data Type	Description
		information, refer to the section Tool Motion — Helical Approach on page 1717. • PRO_TM_TYPE_HELICAL_EXIT. For more information, refer to the section Tool Motion — Helical Exit on page 1720. • PRO_TM_TYPE_RAMP_APPROACH. For more information, refer to the section Tool Motion — Ramp Approach on page 1758. • PRO_TM_TYPE_RAMP_EXIT. Refer to the section Tool Motion — Ramp Exit on page 1760. • PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_ENTRY_PNT_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the entry point selection. Effects start location of the first cut.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step.

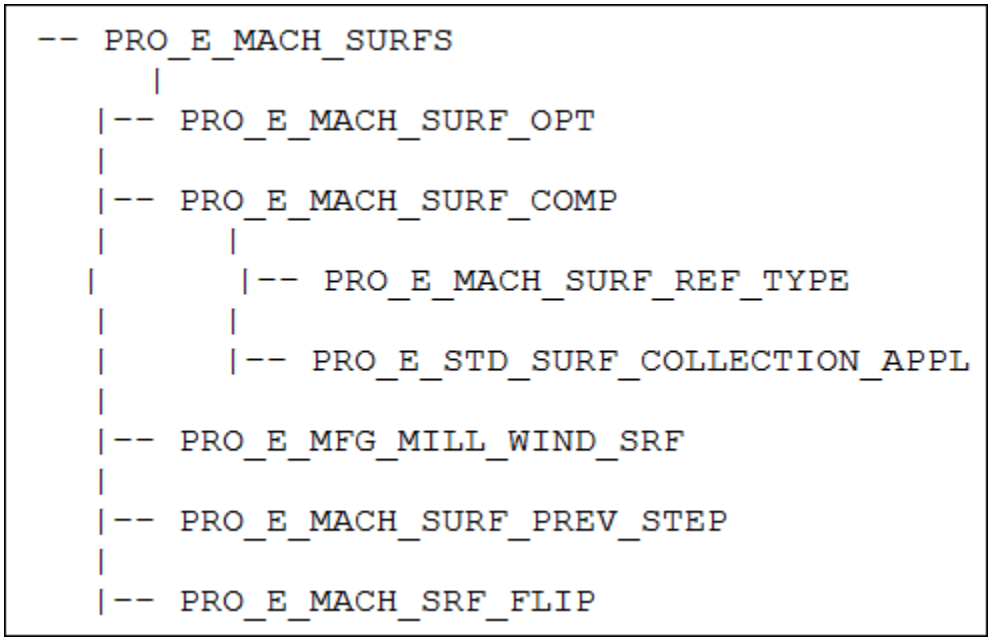
Element ID	Data Type	Description
		Note This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: Machining Surfaces

The Machining Surface Element Tree:

The element tree for the machining surfaces feature is documented in the header file `ProMfgElemMachSurf.h`, and is as shown in the following figure:

Element Tree for Machining Surface Feature



Element ID	Data Type	Description
PRO_E_MACH_SURF_OPT	PRO_VALUE_TYPE_INT	This element specifies the controlling of the object to be selected. The valid values for this element are: • PRO_MACH_SURF_OPT_SURFACES—Specifies that surface collection will be defined. • PRO_MACH_SURF_OPT_MILL_WIND—Specifies that window selection will be defined (for Face Milling). • PRO_MACH_SURF_OPT_PREV_STEP—Specifies that previous step will be defined (not for Face Milling).
PRO_E_MACH_SURF_COMP	Compound	This compound element specifies the collection of surfaces. **Note** This element is ignored if the element PRO_E_MACH_SURF_OPT is not set to PRO_MACH_SURF_OPT_SURFACES.
PRO_E_MACH_SURF_REF_TYPE	PRO_VALUE_TYPE_INTEGER	Specifies the reference types. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_SURF_REF_TYPE_REFPART—Specifies that surfaces from a reference part are collected. • PRO_SURF_REF_TYPE_WORKPIECE—Specifies that surfaces from a workpiece part are collected. • PRO_SURF_REF_TYPE_MILL_VOLUME—Specifies that mill volume is selected or individual surfaces belonging to a mill volume are collected. • PRO_SURF_REF_TYPE_MILL_MVOLSRF—Specifies that a Mill Surface is selected or individual surfaces belonging to a Mill Surface quilt are collected. • PRO_SURF_REF_TYPE_TOP_ASSEM_SRF—Specifies that quilt created at the assembly level is selected.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	This element specifies the collection of selected surfaces to be machined. **Note** This element is mandatory if neither PRO_E_MFG_MILL_WIND_SRF nor PRO_E_MACH_SURF_PREV_STEP are defined.
PRO_E_MFG_MILL_WIND_SRF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of mill window feature. **Note** Specify this element only if the elements PRO_E_STD_SURF_COLLECTION_APPL and PRO_E_MACH_SURF_PREV_STEP are not defined.

Element ID	Data Type	Description
PRO_E_MACH_SURF_PREV_STEP	PRO_VALUE_TYPE_SELECTION	Specifies the selection of an Creo NC step feature. **Note** This element is mandatory if neither the element PRO_E_STD_SURF_COLLECTION_APPL nor the element PRO_E_MFG_MILL_WIND_SRF are defined. This element must not be set otherwise.
PRO_E_MACH_SRF_FLIP	PRO_VALUE_TYPE_INTEGER	Specifies the position of flip quilt. The valid values for this element are: • PRO_B_FALSE—Specifies the machining of the surface on the side defined by the default normal. • PRO_B_TRUE—Specifies the machining of the surface on the opposite side defined by the default normal. **Note** This element is mandatory if the element PRO_E_MACH_SURFS is defined and the reference type is set to PRO_SURF_REF_TYPE_MILL_MVOLSRF or PRO_SURF_REF_TYPE_TOP_ASSEM_SRF.

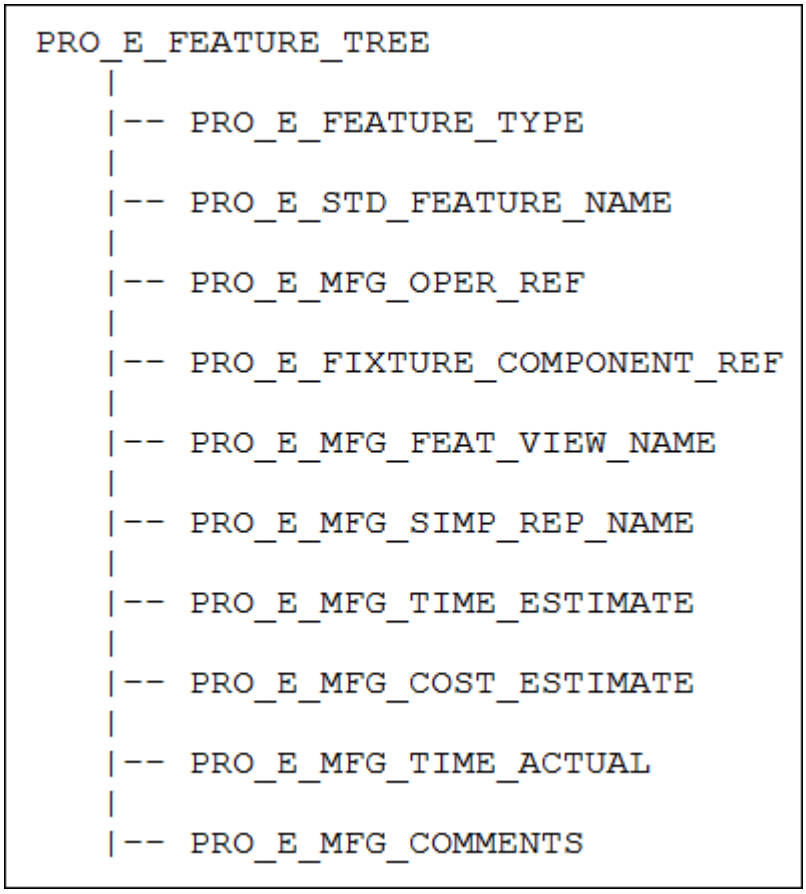
Element Trees: Fixture Definition

This section describes how to construct and access the element tree for Manufacturing Fixture Setup. It also describes how to create, redefine, and access the properties of these features.

The Manufacturing Fixture Setup Element Tree:

The element tree for the manufacturing fixture setup is documented in the header file `ProMfgFeatFixture.h`, and is as shown in the following figure:

Element Tree for Manufacturing Fixture Setup feature



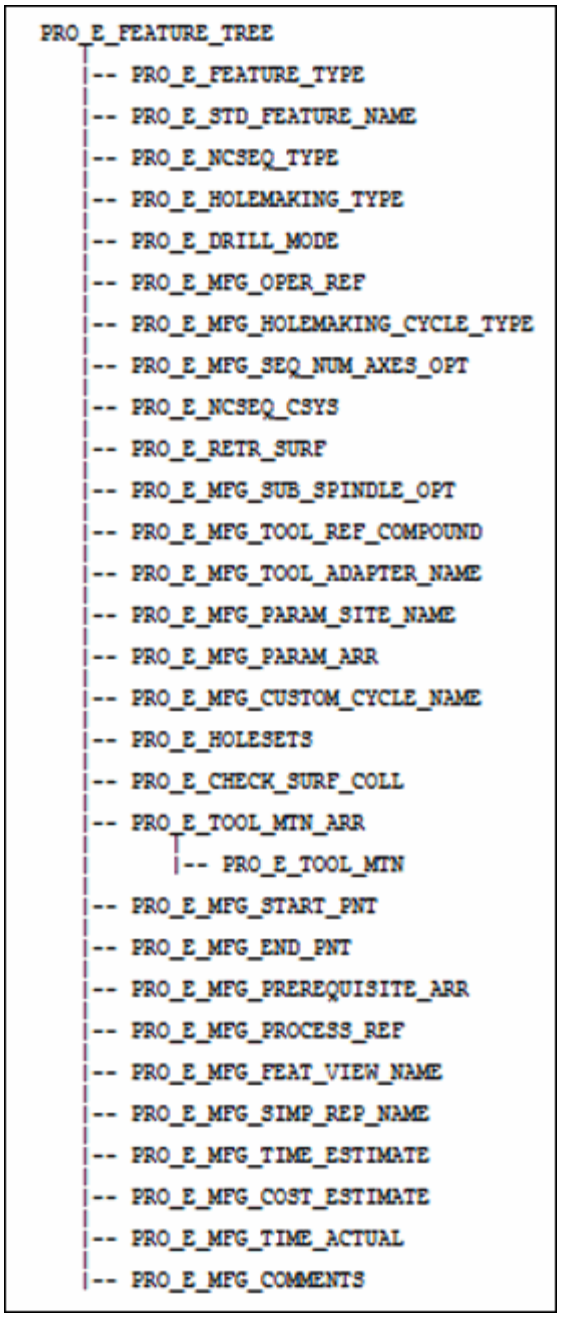
The following table lists the contents of `PRO_E_FEATURE_TREE` element.

Element ID	Data Type	Description
`PRO_E_FEATURE_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies the type of the feature. The valid value for this element is `PRO_FEAT_FIXSETUP`.
`PRO_E_STD_FEATURE_NAME`	`PRO_VALUE_TYPE_WSTRING`	Optional element. Specifies the name for the Creo NC sequence. The default value is `FSETP1`.
`PRO_E_MFG_OPER_REF`	`PRO_VALUE_TYPE_SELECTION`	Mandatory element. Specifies the operation feature selection.
`PRO_E_FIXTURE_COMPONENT_REF`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Specifies the selection of the reference components. This element supports multiple selections. **Note** Specify this element only if the component is inserted in the manufacturing assembly.

Element ID	Data Type	Description
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Manufacturing Holemaking Step

The element tree for PRO_E_FEATURE_TREE is documented in the header file ProMfgFeatHolemaking.h, and is shown in the following figure.



The following table lists the contents of PRO_E_FEATURE_TREE element.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_DRILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the sequence feature name.

Element ID	Data Type	Description
PRO_E_SEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the sequence type. The valid value for this element is PRO_SEQ_HOLEMAKING.
PRO_E_HOLEMAKING_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of holemaking sequence using the enumerated value ProHolemakingType.
PRO_E_DRILL_MODE	PRO_VALUE_TYPE_INT	Specifies the holemaking mode - machining with rotating part or rotating tool using the enumerated value ProDrillModeOption. **Note** This element is mandatory for drilling on mill-turn work center.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_MFG_HOLEMAKING_CYCLE_TYPE	PRO_VALUE_TYPE_INT	Specifies the holemaking cycle type using the enumerated value ProHmCycleType. **Note** This element is mandatory for drilling, tapping, counterboring, countersinking. For countersinking, this element should be set to the value PRO_HM_CYCLE_TYPE_STANDARD. This element is ignored for other types of holemaking.
PRO_E_MFG_SEQ_NUM_AXES_OPT	PRO_VALUE_TYPE_INT	Specifies the number of axes. The valid value for this element are: • 3 • 5 **Note** This element can be set to 5 only if work center allows 5-axis machining.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the coordinate system datum feature to be used as a sequence coordinate system.

Element ID	Data Type	Description
PRO_E_RETR_SURF	Compound	Specifies the retract compound definition. For more information, refer to the section Retract Elements on page 1673 for more information on the element tree. **Note** • This element is mandatory when the element PRO_E_DRILL_MODE is set to PRO_DRILL_HOLE_ON_MILL. • This element is ignored when the element PRO_E_DRILL_MODE is set to PRO_DRILL_HOLE_ON_LATHE.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Specifies the tool reference compound definition. For more information, refer to the section Tool Reference on page 1676 for more information on the element tree.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of site with default values

Element ID	Data Type	Description
		for manufacturing parameters. **Note** The name will be ignored if site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_CUSTOM_CYCLE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of custom cycle which will be used to define machining strategy. **Note** • A custom cycle with specified name must exists in manufacturing model. • This element is mandatory for custom cycle holemaking (when the element PRO_E_HOLEMAKING_TYPE is not set to PRO_HOLE_MK_CUSTOM)
PRO_E_HOLESETS	Array	Specifies an array of holesets which gives specification of holes to machine. For more information, refer to the section Manufacturing Holemaking Holeset on page 1584 for more information on the element tree.
PRO_E_CHECK_SURF_COLL	PRO_VALUE_TYPE_SELECTION	Specifies the check surfaces compound definition. The element tree for the Checking Surfaces is defined in the header file ProMfgElemCheckSurf.h. For more information, refer to the section Checking Surfaces on page 1687 for more information on the element tree.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_ARR	Array	Optional element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Optional element. Specifies the tool motion compound specification. The applicable tool motion types for this element are: • Tool Motion — Auto Cut on page 1766 • Tool Motion — Go To Point on page 1696 • Tool Motion — Go Delta on page 1700 • Tool Motion — Go Home on page 1704 • Tool Motion — Go Retract on page 1708 • Tool Motion — Connect on page 1762 • Tool Motion — CL Command on page 1767 .
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to start machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. Allows to end machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. Specifies the array of ids of prerequisite sequences. The element tree for the Sequence Prerequisites is defined in the header file ProMfgElemPrerequisite.h. For more information, refer to the section Sequence Prerequisites on page 1682 for more information on the element tree.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometry references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the view name. It allows you to

Element ID	Data Type	Description
		associate specific view with a machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. It allows you to associate a specific simplified representation with a machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimate. It allows you to specify time estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate. It allows you to specify cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time. It allows you to specify actual time for the machining step **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Manufacturing Holemaking Holeset

The element PRO_E_HOLESETS specifies an array of holesets and gives specifications about machining references. This element is documented in the header file ProMfgElemHoleset.h, and is shown in the following figure.

Element tree for Manufacturing Holemaking Holeset

```
-- PRO_E_HOLESETS
      |
      |-- PRO_E_HOLESET
            |
            |-- PRO_E_HOLESET_ID
            |
            |-- PRO_E_HOLESET_TYPE
            |
            |-- PRO_E_HOLESET_START
            |
            |-- PRO_E_HOLESET_END
            |
            |-- PRO_E_HOLESET_DEPTH
            |
            |-- PRO_E_HOLESET_CUSTOM_CYCLE_PLATES
            |
            |-- PRO_E_DRILL_PART_DATA
            |       |
            |       |--PRO_E_AUTO_SEL_DRILL_PARTS
            |       |
            |       |--PRO_E_DRILL_PARTS
            |
            |-- PRO_E_AUTODRILL_INFO
            |       |
            |       |--PRO_E_HOLESET_AUTODRILL_REF_INDEX
            |       |
            |       |--PRO_E_AUTODRILL_DEPTH_BY_TABLE
            |
            |-- PRO_E_HOLESET_ORIENT_TYPE
            |
            |-- PRO_E_HOLESET_ORIENT_REF
            |
            |-- PRO_E_HOLESET_TIP_CTRL_PNT
            |
            |-- PRO_E_HOLESET_SELECTION_RULES
            |
            |-- PRO_E_MFG_HSET_START_HOLE_REF
            |
            |-- PRO_E_HOLESET_UNFLIPPED_AXES
            |
            |-- PRO_E_HOLESET_FLIPPED_AXES
            |
            |-- PRO_E_HOLESET_GANG_TOOL_INFO
                  |
                  |--PRO_E_HSET_GANG_TOOL_PARENT_ID
                  |
                  |--PRO_E_HSET_IS_GANG_TOOL_LEADER
```

The following table lists the contents of PRO_E_HOLESETS element.

Element ID	Data Type	Description
PRO_E_HOLESET	Compound	Mandatory element. Specifies the holeset definition. **Note** The holeset array must have at least one member.
PRO_E_HOLESET_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the holeset type - axes, points or geometry. The values for this element are defined by ProHolesetType.
PRO_E_HOLESET_START	Compound	Mandatory element. Specifies the holemaking start compound specification. **Note** This element is ignored for web drilling. Mandatory for other types of holemaking.
PRO_E_HOLESET_END	Compound	Specifies the holemaking depth compound specification. **Note** This element is ignored for web drilling. Mandatory for other types of holemaking.
PRO_E_HOLESET_DEPTH	Compound	Specifies the web drilling depth compound specification. **Note** Mandatory for web drilling. Ignored for other types of holemaking.
PRO_E_HOLESET_CUSTOM_CYCLE_PLATES	Compound	Specifies the compound definition of the custom cycle. **Note** This element is mandatory for custom cycle holemaking. Ignored for other types of holemaking.
PRO_E_DRILL_PART_DATA	Compound	Specifies the compound information about components used in depth computation.

Element ID	Data Type	Description
		Mandatory element if start or end of machining has to be computed and at least one of the following hold true: • PRO_E_HOLESET_START_TYPE is set to PRO_DRILL_AUTO_START. • PRO_E_HOLESET_END_TYPE is set to PRO_DRILL_AUTO or PRO_DRILL_THRU_ALL. • PRO_E_HOLESET_DEPTH_TYPE is set to PRO_DRILL_AUTO or PRO_DRILL_THRU_ALL.
PRO_E_AUTO_SEL_DRILL_PARTS	PRO_VALUE_TYPE_INT	Mandatory element. The option defines the way components are collected. The valid values for this element are: • TRUE—All components of type "reference part" or "workpiece" are considered for depth calculation. • FALSE—Only selected components are considered in depth calculation.
PRO_E_DRILL_PARTS	PRO_VALUE_TYPE_SELECTION	Specifies the components selections. This element supports multiple selections. **Note** Mandatory element if PRO_E_AUTO_SEL_DRILL_PARTS is set to FALSE.
PRO_E_AUTODRILL_INFO	Compound	A compound element specifying auto drilling information. **Note** This element is mandatory for auto drilling hole sets.
PRO_E_AUTODRILL_DEPTH_BY_TABLE	PRO_VALUE_TYPE_INT	This element is used for auto drilling hole sets. The valid values for this element are: • TRUE—It reads the depth information from the auto drilling table. • FALSE—It takes the depth information from the hole set

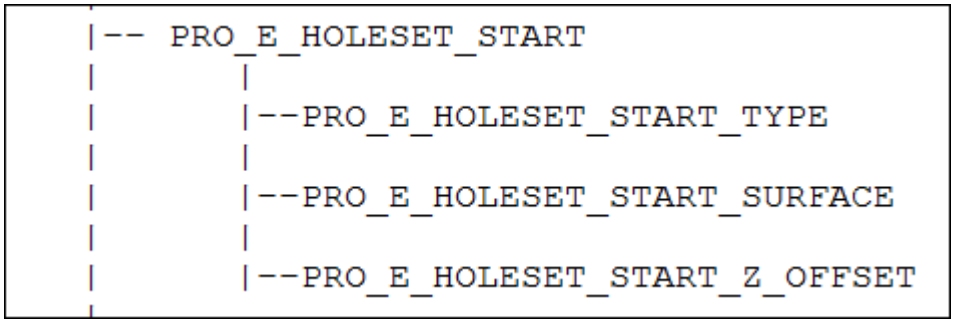
Element ID	Data Type	Description
		definition. **Note** This element is mandatory for auto drilling hole sets.
PRO_E_HOLESET_ORIENT_TYPE	PRO_VALUE_TYPE_INT	Specifies the machining direction option for 5-axis holemaking from the selected reference. The valid values for this element are: • Away • Toward The values for this element are defined by ProDrillOrientType. **Note** This element is optional for axes holeset for 5-axis holemaking. Ignored in all other cases.
PRO_E_HOLESET_ORIENT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the single reference selection (surface, point, axis) to define the direction of machining for 5-axis holemaking.The valid values for this element are: • Away—Applicable when the element PRO_E_HOLESET_ORIENT_TYPE is set to PRO_DRILL_ORIENT_TO_REF. • Toward—Applicable when the element PRO_E_HOLESET_ORIENT_TYPE is set to PRO_DRILL_ORIENT_FROM_REF **Note** This element is optional for axes holeset for 5-axis holemaking. Ignored in all other cases.
PRO_E_HOLESET_TIP_CTRL_PNT	PRO_VALUE_TYPE_INT	Optional element. Specifies the cutting tool control tip number. It defines a point on the cutting tool to be controlled during machining

Element ID	Data Type	Description
		of the hole set. **Note** The first point of the tool is used if the element does not exist.
PRO_E_HOLESET_SELECTION_RULES	Compound	Mandatory element. Specifies the compound information about location of holes to drill. **Note** Define this element when at least one of the following child elements are defined: • PRO_E_HOLESET_SEL_PNTS_ON_SURFACES • PRO_E_HOLESET_SEL_INDIV_PNTS • PRO_E_HOLESET_SEL_PNTS_BY_FEATURE • PRO_E_HOLESET_SEL_AUTO_CHAMFER • PRO_E_HOLESET_SEL_INDIV_AXES • PRO_E_HOLESET_SEL_AXIS_PATTS • PRO_E_MFG_HSET_DRILL_GROUP_REF • PRO_E_HOLESET_SEL_BY_SURFACES • PRO_E_MFG_HSET_DIAM_TYPE_OPT • PRO_E_MFG_HSET_DIAM_ARR • PRO_E_MFG_HSET_PARAM_ARR
PRO_E_MFG_HSET_START_HOLE_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the axis selection (for axes holeset) or

Element ID	Data Type	Description
		point selection (for points holeset) for the hole to be drilled first.
PRO_E_HOLESET_UNFLIPPED_AXES	PRO_VALUE_TYPE_SELECTION	Specifies the axis selections for holes that should be drilled as per orientation of the axis entities. This element supports multiple selections. **Note** This element is optional for axes holeset for 5-axis holemaking. Ignored in all other cases.
PRO_E_HOLESET_FLIPPED_AXES	PRO_VALUE_TYPE_SELECTION	Specifies the axis selections for holes that should be drilled in the direction opposite to the orientation of the axis entities. This element supports multiple selections. **Note** This element is optional for axes holeset for 5-axis holemaking. Ignored in all other cases.
PRO_E_HOLESET_GANG_TOOL_INFO	Compound	Optional element. A compound element specifying the gang tool properties.
PRO_E_HSET_GANG_TOOL_PARENT_ID	PRO_VALUE_TYPE_INT	This element is mandatory for gang tool specification. Specifies the parent step id.
PRO_E_HSET_IS_GANG_TOOL_LEADER	PRO_VALUE_TYPE_INT	This element is mandatory for gang tool specification. Specifies if holeset is a leader of the gang tool. It takes the following values: • True—Specifies that the holeset is a leader of the gang tool. • False—Specifies that the holeset is not a leader of the gang tool.

Element tree for PRO_E_HOLESET_START

The element tree for PRO_E_HOLESET_START is as shown in the figure below:



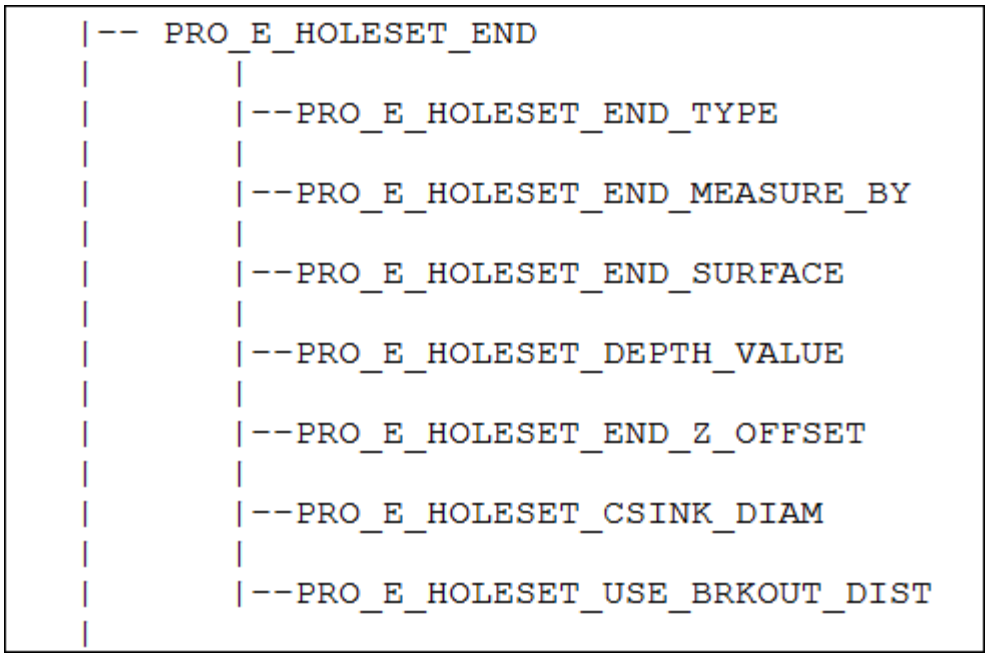
The following table lists the sub elements of the element `PRO_E_HOLESET_START` defined in the header file `ProMfgElemHoleset.h`.

Element ID	Data Type	Description
`PRO_E_HOLESET_START`	`Compound`	Mandatory element. Specifies the holemaking start compound specification. **Note** This element is ignored for web drilling. Mandatory for other types of holemaking.
`PRO_E_HOLESET_START_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies the start surface option using the enumerated value `ProDrillStartType`. **Note** This element is ignored for web drilling and counersinking (when `PRO_E_HOLEMAKING_TYPE` is set to`PRO_HOLE_MK_CSINK` or `PRO_HOLE_MK_WEB`). Mandatory for other types of holemaking.

Element ID	Data Type	Description
PRO_E_HOLESET_START_SURFACE	PRO_VALUE_TYPE_SELECTION	Specifies the starting surface or quilt selection. **Note** • For countersink, this element is optional if auto_chamfer rule is set and there is no explicit axes selection. • For web drilling this element is ignored. • For other holemaking types this element is mandatory if PRO_E_HOLESET_START_TYPE is set to PRO_DRILL_FROM_SURFACE.
PRO_E_HOLESET_START_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Specifies that the drilling will start at this offset from sequence coordinate system origin. **Note** This element is mandatory if the element PRO_E_HOLESET_START_TYPE is set to PRO_DRILL_START_OFFSET_FROM_CSYS. Ignored in all other cases.

Element tree for PRO_E_HOLESET_END

The element tree for PRO_E_HOLESET_END is as shown in the figure below:



The following table lists the sub elements of the element `PRO_E_HOLESET_END` defined in the header file `ProMfgElemHoleset.h`.

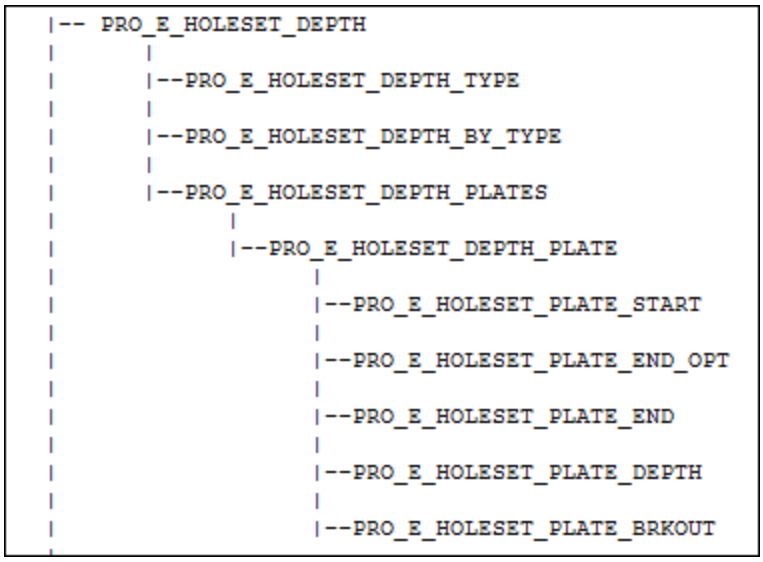
Element ID	Data Type	Description
`PRO_E_HOLESET_END`	`Compound`	Specifies the holemaking depth compound specification. **Note** This element is ignored for web drilling. Mandatory for other types of holemaking.
`PRO_E_HOLESET_END_TYPE`	`PRO_VALUE_TYPE_INT`	Specifies the end type option. The values for this element are defined by `ProDrillEndType`. **Note** This element is ignored for counersinking. Mandatory for other types of holemaking.
`PRO_E_HOLESET_END_MEASURE_BY`	`PRO_VALUE_TYPE_INT`	Specifies the depth calculation option.The valid values for this element are • Tip • Shoulder

Element ID	Data Type	Description
		The values for this element are defined by `ProDrillDepthByType`. **Note** This element is ignored for counersinking and web drilling. Mandatory for other types of holemaking.
`PRO_E_HOLESET_END_SURFACE`	`PRO_VALUE_TYPE_SELECTION`	Specifies the end surface or quilt selection. **Note** • This element is ignored for countersinking and web drilling. • This element is ignored for custom drilling if cycle definition does not use end surface. • This element is mandatory if `PRO_E_HOLESET_END_TYPE` is set to `PRO_DRILL_UPTO_SURFACE`. Ignored in all other cases.
`PRO_E_HOLESET_DEPTH_VALUE`	`PRO_VALUE_TYPE_DOUBLE`	Specifies the depth to drill from start. **Note** This element is mandatory if `PRO_E_HOLESET_END_TYPE` is set to `PRO_DRILL_OFFSET_FROM_START`. Ignored in all other cases.
`PRO_E_HOLESET_END_Z_OFFSET`	`PRO_VALUE_TYPE_DOUBLE`	Specifies that the drilling will end at this offset from sequence

Element ID	Data Type	Description
		coordinate system origin. **Note** This element is mandatory if the element PRO_E_HOLESET_END_TYPE is set to PRO_DRILL_END_OFFSET_FROM_CSYS. Ignored in all other cases.
PRO_E_HOLESET_CSINK_DIAM	PRO_VALUE_TYPE_DOUBLE	Specifies the countersink diameter. **Note** This element is mandatory for countersink sequence (when PRO_E_HOLEMAKING_TYPE is set to PRO_HOLE_MK_CSINK). Ignored in all other cases.
PRO_E_HOLESET_USE_BRKOUT_DIST	PRO_VALUE_TYPE_DOUBLE	Optional element.Specifies the depth breakout option. The valid values for this element are: • TRUE—To drill an additional BREAKOUT_DISTANCE (manufacturing parameter) deeper. • FALSE— Does not drill deeper than defined by depth specification. Ignored for countersink sequences. **Note** This element is ignored if end_type is PRO_DRILL_THRU_ALL_PARTS or PRO_DRILL_AUTO_END.

Element tree for PRO_E_HOLESET_DEPTH

The element tree for PRO_E_HOLESET_DEPTH is as shown in the figure below:



The following table lists the sub elements of the element `PRO_E_HOLESET_DEPTH` defined in the header file `ProMfgElemHoleset.h`.

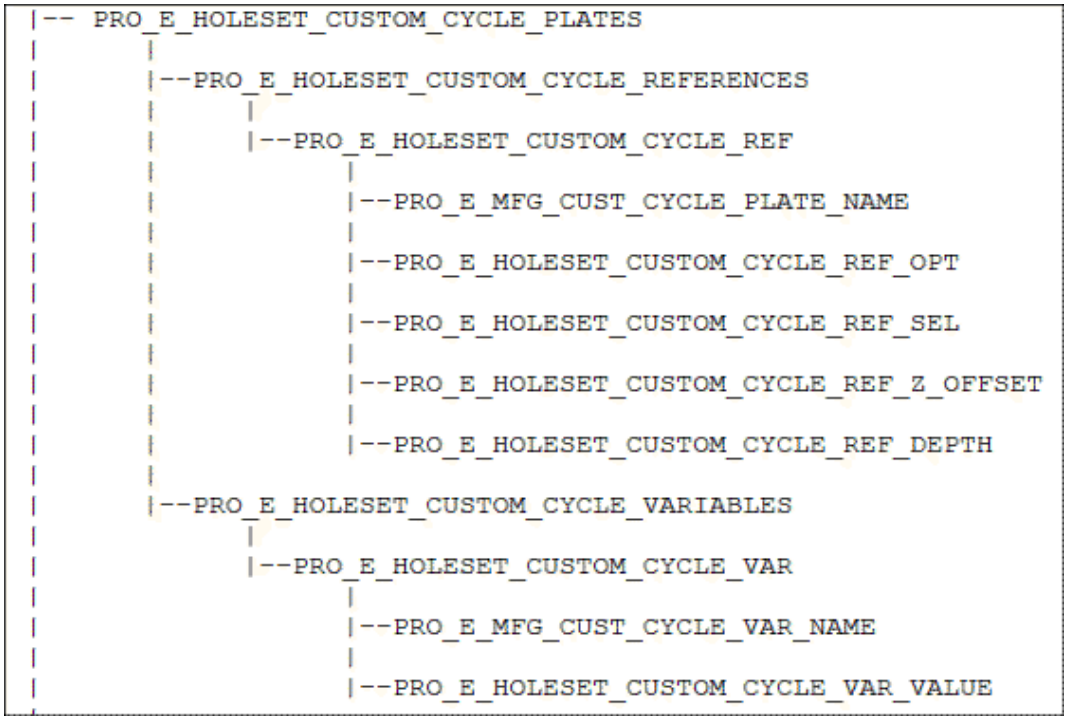
Element ID	Data Type	Description
`PRO_E_HOLESET_DEPTH`	`Compound`	Specifies the web drilling depth compound specification. **Note** Mandatory for web drilling. Ignored for other types of holemaking.
`PRO_E_HOLESET_DEPTH_TYPE`	`PRO_VALUE_TYPE_INT`	Specifies the web drilling depth type option. The valid values for this element are: • Blind • Auto • Through • The values for this element are defined by `ProDrillDepthType`. **Note** This element is mandatory for web drilling. Ignored for other types of holemaking.
`PRO_E_HOLESET_DEPTH_BY_TYPE`	`PRO_VALUE_TYPE_INT`	Specifies the web drilling depth calculation option. The valid values for this element are: • Tip • Shoulder • The values for this element are

Element ID	Data Type	Description
		defined by ProDrillDepthByType. **Note** This element is mandatory for web drilling. Ignored for other types of holemaking.
PRO_E_HOLESET_DEPTH_PLATES	Array	Specifies an array of web drilling plates. **Note** This element is mandatory for web drilling. Ignored for other types of holemaking.
PRO_E_HOLESET_DEPTH_PLATE	Compound	Mandatory element. Specifies the web drilling plate definition. **Note** At least one plate must be defined in array of plates.
PRO_E_HOLESET_PLATE_START	PRO_VALUE_TYPE_SELECTION	Specifies the web drilling plate starting surface or quilt selection. **Note** This element is mandatory if PRO_E_HOLESET_DEPTH_TYPE is set to PRO_DRILL_BLIND. Ignored in all other cases.
PRO_E_HOLESET_PLATE_END_OPT	PRO_VALUE_TYPE_INT	Specifies the web drilling plate depth type option. The values for this element are defined by ProDrillEndType. **Note** This element is mandatory for web drilling. Ignored for other types of holemaking.
PRO_E_HOLESET_PLATE_END	PRO_VALUE_TYPE_SELECTION	Specifies the web drilling plate

Element ID	Data Type	Description
		end surface or quilt selection. **Note** This element is mandatory for web drilling if PRO_E_HOLESET_PLATE_END_OPT is set to PRO_DRILL_UPTO_SURFACE. Ignored in all other cases.
PRO_E_HOLESET_PLATE_DEPTH	PRO_VALUE_TYPE_DOUBLE	Specifies the web drilling depth from plate start. **Note** Mandatory for web drilling if PRO_E_HOLESET_PLATE_END_OPT is set to PRO_DRILL_OFFSET_FROM_START. Ignored in all other cases.
PRO_E_HOLESET_PLATE_BRKOUT	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the web drilling depth breakout option. The valid values for this element are: • TRUE—Drills an additional BREAKOUT_DISTANCE (manufacturing parameter) deeper. • FALSE—Does not drill deeper than defined by depth specification. **Note** This element is ignored for countersink sequences.

Element Tree for PRO_E_HOLESET_CUSTOM_CYCLE_PLATES

The element tree for PRO_E_HOLESET_CUSTOM_CYCLE_PLATES is as shown in the figure below:



The following table lists the sub elements of the element `PRO_E_HOLESET_CUSTOM_CYCLE_PLATES` defined in the header file `ProMfgElemHoleset.h`.

Element ID	Data Type	Description
`PRO_E_HOLESET_CUSTOM_CYCLE_PLATES`	`Compound`	Specifies the compound definition of the custom cycle. **Note** This element is mandatory for custom cycle holemaking. Ignored for other types of holemaking.
`PRO_E_HOLESET_CUSTOM_CYCLE_REFERENCES`	`Array`	Specifies an array of custom cycle

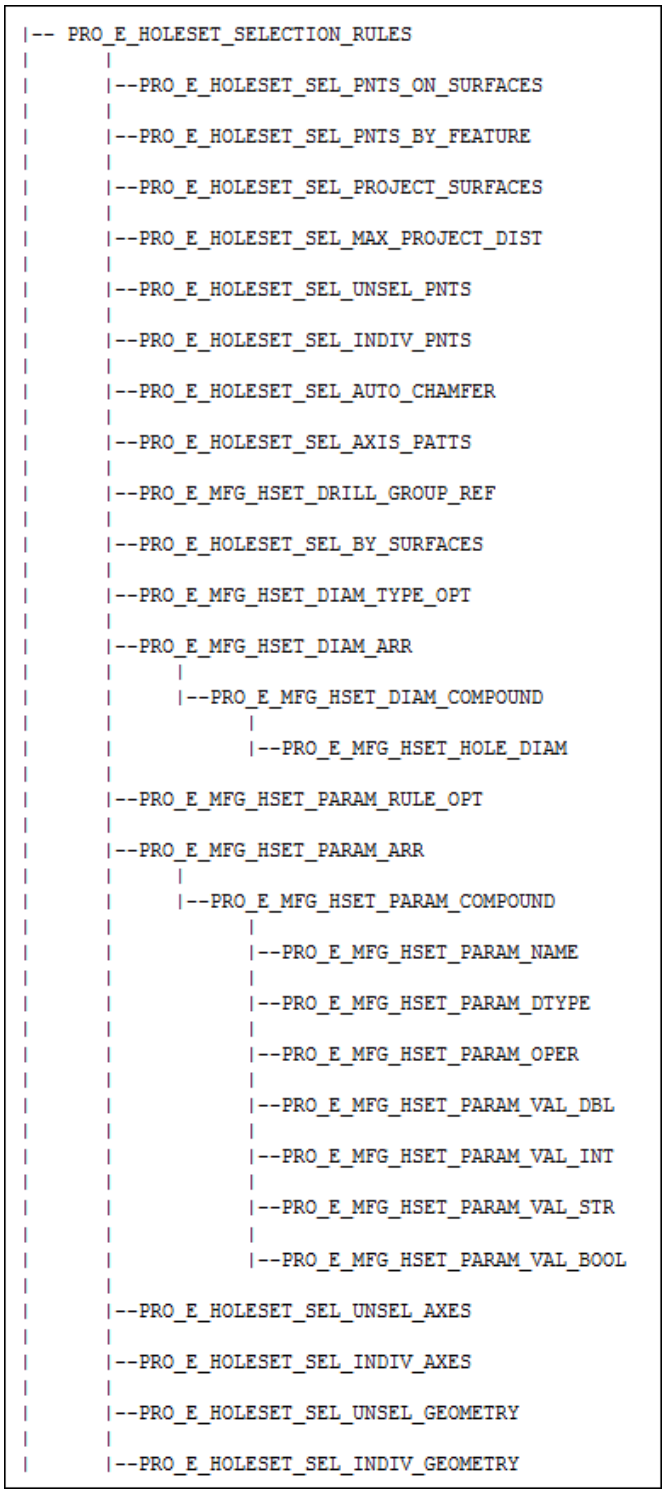
Element ID	Data Type	Description
		references. **Note** • The number of members in array should match with number of references in custom cycle definition (stored in manufacturing model). • Mandatory if references are used in the custom cycle definition (see PRO_E_MFG_CUSTOM_CYCLE_NAME).
PRO_E_HOLESET_CUSTOM_CYCLE_REF	Compound	Mandatory element for custom cycle with references. Specifies the compound definition of a custom cycle reference.
PRO_E_MFG_CUST_CYCLE_PLATE_NAME	Compound	Mandatory element for reference definition. Specifies the name of custom cycle reference.
PRO_E_HOLESET_CUSTOM_CYCLE_REF_OPT	PRO_VALUE_TYPE_INT	Mandatory element for reference definition. Specifies the type of reference specification. The values for this element are defined by ProCustomRefOption.
PRO_E_HOLESET_CUSTOM_CYCLE_REF_SEL	PRO_VALUE_TYPE_SELECTION	Specifies the custom cycle reference surface or quilt selection. **Note** Mandatory if PRO_E_HOLESET_CUSTOM_CYCLE_REF_OPT is set to PRO_CUSTOM_DRILL_SELECT_REFERENCE. Ignored in all other cases.
PRO_E_HOLESET_CUSTOM_CYCLE_REF_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Specifies the custom cycle reference specification by offset value from sequence coordinate

Element ID	Data Type	Description
		system origin. **Note** Mandatory if PRO_E_HOLESET_CUSTOM_CYCLE_REF_OPT is set to PRO_CUSTOM_DRILL_OFFSET_FROM_CSYS. Ignored in all other cases.
PRO_E_HOLESET_CUSTOM_CYCLE_REF_DEPTH		Specifies the custom cycle reference specification by offset value from start surface. **Note** Mandatory if PRO_E_HOLESET_CUSTOM_CYCLE_REF_OPT is set to PRO_CUSTOM_DRILL_OFFSET_FROM_START. Ignored in all other cases.
PRO_E_HOLESET_CUSTOM_CYCLE_VARIABLES	Array	Specifies an array of custom cycle variables. **Note** • The number of members in array should match with number of variables in the custom cycle definition (stored in manufacturing model). • Mandatory if variables are used in named custom cycle (see PRO_E_MFG_CUSTOM_CYCLE_NAME).
PRO_E_HOLESET_CUSTOM_CYCLE_VAR	Compound	Mandatory element for custom cycle with variables. Specifies the compound definition of a custom cycle variable.

Element ID	Data Type	Description
PRO_E_MFG_CUST_CYCLE_VAR_NAME	PRO_VALUE_TYPE_WSTRING	Mandatory element for variable definition. Specifies the name of custom cycle variable.
PRO_E_HOLESET_CUSTOM_CYCLE_VAR_VALUE	PRO_VALUE_TYPE_DOUBLE	Mandatory element for variable definition. Specifies the Custom cycle variable value.

Element tree for PRO_E_HOLESET_SELECTION_RULES

The element tree for PRO_E_HOLESET_SELECTION_RULES is as shown in the figure below:



The following table lists the sub elements of the element `PRO_E_HOLESET_SELECTION_RULES` defined in the header file `ProMfgElemHoleset.h`.

Element ID	Data Type	Description
PRO_E_HOLESET_ SELECTION_RULES	Compound	Mandatory element. Specifies the compound information about location of holes to drill. **Note** Define this element when at least one of the following child elements are defined: • PRO_E_HOLESET_ SEL_PNTS_ON_ SURFACES • PRO_E_HOLESET_ SEL_INDIV_PNTS • PRO_E_HOLESET_ SEL_PNTS_BY_ FEATURE • PRO_E_HOLESET_ SEL_AUTO_CHAMFER • PRO_E_HOLESET_ SEL_INDIV_AXES • PRO_E_HOLESET_ SEL_AXIS_PATTS • PRO_E_MFG_HSET_ DRILL_GROUP_REF • PRO_E_HOLESET_ SEL_BY_SURFACES • PRO_E_MFG_HSET_ DIAM_TYPE_OPT • PRO_E_MFG_HSET_ DIAM_ARR • PRO_E_MFG_HSET_ PARAM_ARR
PRO_E_HOLESET_SEL_PNTS_ ON_SURFACES	PRO_VALUE_TYPE_ SELECTION	Specifies the selection of surfaces (or quilts) with points. This element supports multiple

Element ID	Data Type	Description
		selections. **Note** • This element is optional for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS). • This element is ignored for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES).
PRO_E_HOLESET_SEL_PNTS_BY_FEATURE	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum point features. This element supports multiple selections. **Note** • This element is optional element for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS). • This element is ignored for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES)
PRO_E_HOLESET_SEL_PROJECT_SURFACES	PRO_VALUE_TYPE_SELECTION	Specifies the selection of surfaces or quilts. This element supports

Element ID	Data Type	Description
		multiple selections. **Note** • This element is mandatory for five-axis holemaking (PRO_E_MFG_SEQ_NUM_AXES_OPT is set to 5) if at least one point selection is set for one of the following elements: ○ PRO_E_HOLESET_SEL_INDIV_PNTS ○ PRO_E_HOLESET_SEL_PNTS_BY_FEATURE • Only the points placed on selected surface will be considered for machining in the direction normal to the surface (at the location defined by the point). • This element is ignored for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES).
PRO_E_HOLESET_SEL_MAX_PROJECT_DIST	PRO_VALUE_TYPE_DOUBLE	Specifies the accuracy used to determine whether a point lies on a

Element ID	Data Type	Description
		surface. **Note** • This element is mandatory for five-axis holemaking (PRO_E_MFG_SEQ_NUM_AXES_OPTis set to 5) if at least one point selection is set for one of the following elements: ○ PRO_E_HOLESET_SEL_INDIV_PNTS ○ PRO_E_HOLESET_SEL_PNTS_BY_FEATURE • Only the points placed on selected surface will be considered for machining in the direction normal to the surface (at the location defined by the point). • This element is ignored for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES).
PRO_E_HOLESET_SEL_UNSEL_PNTS	PRO_VALUE_TYPE_SELECTION	Specifies the selection of points for holes to be excluded for machining. This element supports

Element ID	Data Type	Description
		multiple selections. **Note** • This element is optional for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS). • This element is ignored for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES).
PRO_E_HOLESET_SEL_INDIV_PNTS	PRO_VALUE_TYPE_SELECTION	Specifies the selection of points. This element supports multiple selections. **Note** • This element is mandatory for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS) if none of the following elements is defined: ○ PRO_E_HOLESET_SEL_PNTS_ON_SURFACES ○ PRO_E_HOLESET_SEL_PNTS_BY_FEATURE • Ignored for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES).
PRO_E_HOLESET_SEL_AUTO_CHAMFER	PRO_VALUE_TYPE_INT	Specifies the auto chamfer option. The valid values for this element are:

Element ID	Data Type	Description
		• TRUE—Collects holes with chamfers matching chamfer angle with angle of cutting tool (see PRO_E_MFG_TOOL_REF_COMPOUND for tool reference). • FALSE—Does not collect holes by chamfer dimensions. **Note** This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_HOLESET_SEL_AXIS_PATTS	PRO_VALUE_TYPE_SELECTION	Specifies the selection of axes of patterned holes. If a pattern leader is selected, all holes in pattern will be collected. This element supports multiple selections. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_MFG_HSET_DRILL_GROUP_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of drill group features. This element

Element ID	Data Type	Description
		supports multiple selections. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_HOLESET_SEL_BY_SURFACES	PRO_VALUE_TYPE_SELECTION	Specifies the selection of surfaces or quilts with holes. This element supports multiple selections. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_MFG_HSET_DIAM_TYPE_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of hole diameters that can be collected in the element PRO_E_MFG_HSET_DIAM_ARR. The type of hole diameter is specified using the enumerated data type ProHolesetDiamType. The valid values are:

Element ID	Data Type	Description
		• PRO_HSET_ALL_DIAMS—This is the default value. Specifies that diameters of both solid surfaces and cosmetic threads can be collected. **Note** If the element PRO_E_MFG_HSET_DIAM_TYPE_OPT is not defined, then by default, the hole diameter of type PRO_HSET_ALL_DIAMS is used. • PRO_HSET_HOLE_DIAMS—Specifies that diameters only of solid surfaces can be collected. • PRO_HSET_THREAD_DIAMS—Specifies that diameters only of cosmetic threads can be collected. **Note** This element is not required for holeset of type points and geometry.
PRO_E_MFG_HSET_DIAM_ARR	Array	Specifies an array of diameters of holes to machine. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_MFG_HSET_DIAM_COMPOUND	Compound	Specifies the compound definition

Element ID	Data Type	Description
		of a hole diameter. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_MFG_HSET_HOLE_DIAM	PRO_VALUE_TYPE_DOUBLE	Specifies the diameter of a hole to machine. **Note** This element is a mandatory child of PRO_E_MFG_HSET_DIAM_COMPOUND element.
PRO_E_MFG_HSET_PARAM_RULE_OPT	PRO_VALUE_TYPE_INT	Specifies the type of query that must be used to search for holes to machine. The query type is specified using the enumerated data type ProHsetParamRuleOpt. The valid values are: • PRO_HSET_BOOL_OPER_OR—Collects holes that satisfy at least one of the search conditions set for a parameter. PRO_HSET_BOOL_OPER_AND—Collects holes that satisfy all the search conditions set for a parameter. The search conditions and parameters are defined in the elements PRO_E_MFG_HSET_PARAM*.
PRO_E_MFG_HSET_PARAM_ARR	Array	Specifies an array of search conditions to collect holes for

Element ID	Data Type	Description
		machining. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_MFG_HSET_PARAM_COMPOUND	Compound	Optional element. Specifies a compound element that defines a search condition to match with the user defined parameters in hole features. Each condition defines an expression with user defined parameter name on the left side of the expression and value to compare on the right side. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS).
PRO_E_MFG_HSET_PARAM_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of user defined parameter. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element.

Element ID	Data Type	Description
PRO_E_MFG_HSET_PARAM_DTYPE	PRO_VALUE_TYPE_INT	Specifies the data type of the value. The values for this element are defined by defined by ProParamvalueType. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element.
PRO_E_MFG_HSET_PARAM_OPER	PRO_VALUE_TYPE_INT	Specifies the Ttype of expression operator. The values for this element are defined by defined by ProDrillParamOper. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element.
PRO_E_MFG_HSET_PARAM_VAL_DBL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of type double. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for double data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_DOUBLE). Ignored for other data types.
PRO_E_MFG_HSET_PARAM_VAL_INT	PRO_VALUE_TYPE_INT	Specifies the value of type integer. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for double data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_INTEGER). Ignored for other data types.

Element ID	Data Type	Description
PRO_E_MFG_HSET_PARAM_VAL_STR	PRO_VALUE_TYPE_WSTRING	Specifies the value of type string. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for double data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_STRING). Ignored for other data types.
PRO_E_MFG_HSET_PARAM_VAL_BOOL	PRO_VALUE_TYPE_INT	Specifies the value of type ProBoolean. **Note** This element is a mandatory child of PRO_E_MFG_HSET_PARAM_COMPOUND element for double data type (PRO_E_MFG_HSET_PARAM_DTYPE is set to PRO_PARAM_BOOLEAN). Ignored for other data types.
PRO_E_HOLESET_SEL_UNSEL_AXES	PRO_VALUE_TYPE_SELECTION	Specifies the selection of axes of holes to be excluded for machining. This element supports multiple selections. **Note** • This element is Optional for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES). • This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS). This element supports multiple selections.
PRO_E_HOLESET_SEL_INDIV_AXES	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum axes. Mandatory element for axes holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_AXES) if none of the following elements is defined:

Element ID	Data Type	Description
		• PRO_E_HOLESET_SEL_AUTO_CHAMFER • PRO_E_HOLESET_SEL_AXIS_PATTS • PRO_E_MFG_HSET_DRILL_GROUP_REF • PRO_E_HOLESET_SEL_BY_SURFACES • PRO_E_MFG_HSET_DIAM_ARR • PRO_E_MFG_HSET_PARAM_ARR **Note** This element is ignored for points holeset (PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_POINTS). This element supports multiple selections.
PRO_E_HOLESET_SEL_UNSEL_GEOMETRY	PRO_VALUE_TYPE_SELECTION	Specifies the selection of holes to be excluded for machining. This element supports multiple selections. **Note** • This element is optional for a geometry holeset that is, if the element PRO_E_HOLESET_TYPE is set to PRO_HOLESET_DRILL_GEOM. • This element is ignored for points holeset and axes holeset.

Element ID	Data Type	Description
PRO_E_HOLESET_SEL_INDIV_GEOMETRY	PRO_VALUE_TYPE_SELECTION	Specifies the selection for individual holes. This element supports multiple selections. This element is mandatory for geometry holeset, if none of the following elements are defined: • PRO_E_HOLESET_SEL_AUTO_CHAMFER • PRO_E_HOLESET_SEL_BY_SURFACES • PRO_E_MFG_HSET_DIAM_ARR **Note** This element is ignored for points holeset and axes holeset.

Shut off Surface Feature Element Tree

The shut off tool creates a surface that caps an opening in a molded part. In the Mold and Cast Design mode, the shut off tool creates an assembly surface that references the quilts, surfaces and edges of a part. In the Part design mode, the shut off tool creates part surfaces that reference solid surfaces and quilt surfaces with single-sided or double-sided edges.

The element tree for the shut off surface feature is documented in the header file `ProMoldShutSrf.h`, and is as shown in the following figure:

Element Tree for Mold Shut Off feature:

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_SHUT_SRF_REF_TYPE
    |
    |-- PRO_E_SHUT_SRF_SRF_REFS
    |     |
    |     |-- PRO_E_STD_SURF_COLLECTION_APPL
    |
    |-- PRO_E_SHUT_SRF_COLL_CRV_CMP
    |     |
    |     |-- PRO_E_SHUT_SRF_ARR_COLL_CRV
    |           |
    |           |--PRO_E_STD_CURVE_COLLECTION_APPL
    |
    |-- PRO_E_SHUT_SRF_CAP_SRFS
    |     |
    |     |-- PRO_E_STD_SURF_COLLECTION_APPL
    |
    |-- PRO_E_SHUT_SRF_CLOSE_ALL
```

The following table describes the elements in the element tree for the shut-off surface feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the feature. The valid value for this element is PRO_MOLD_SHUTOFF_SRF.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the feature.
PRO_E_SHUT_SRF_REF_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of references to be selected for the shut off surface. Select either surface references or curve references. The valid values for this element are defined in the enumerated type ProShutSrfRefType: • PRO_SHUT_SRF_SRF_REF— Specifies the references for a surface. • PRO_SHUT_SRF_CRV_REF— Specifies the references

Element ID	Data Type	Description
		for a curve.
PRO_E_SHUT_SRF_SRF_REFS	Compound	Mandatory element. Specifies a compound element that contains the surface references.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Specifies the surfaces for which shut off surfaces need to be created. Select a single surface or multiple surfaces, if the reference type PRO_E_SHUT_SRF_REF_TYPE is set as PRO_SHUT_SRF_SRF_REF. If PRO_E_SHUT_SRF_REF_TYPE is set as PRO_SHUT_SRF_CRV_REF, then all the surfaces are automatically selected.
PRO_E_SHUT_SRF_COLL_CRV_CMP	Compound	Specifies a compound element for curve chains. Chains enable you to perform operations on all of the selected curves in that chain or on multiple chains simultaneously by either including or excluding them from creating shut off surfaces.
PRO_E_SHUT_SRF_ARR_COLL_CRV	Array	Specifies an array of curve chain loops that can be included or excluded from creating shut off surfaces. By default all the selected surfaces are included while creating the shut off surfaces. **Note** The selected loops are excluded while creating shut off surfaces only if the element PRO_E_SHUT_SRF_CLOSE_ALL is set to PRO_SHUT_SRF_SHUT_ALL and this element becomes optional.
PRO_E_STD_CURVE_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Specifies the reference chain loops to be included or excluded while creating shut off surfaces. Select both open and closed chain loops as reference. By default all the selected reference chain loops are included while creating the shut off surfaces. The reference chain loops are excluded only if the value of the element PRO_E_SHUT_SRF_CLOSE_ALL is set

Element ID	Data Type	Description
		to the value `True`. **Note** You can specify only one-sided edges as curve chain loops if the element `PRO_E_SHUT_SRF_REF_TYPE` is set to `PRO_SHUT_SRF_CRV_REF`.
`PRO_E_SHUT_SRF_CAP_SRFS`	`Compound`	Optional element. Specifies a compound element for cap surface references.
`PRO_E_STD_SURF_COLLECTION_APPL`	`PRO_VALUE_TYPE_SELECTION`	Specifies the selection of reference surfaces for capping open loops. This is an optional element and you can select single or multiple surfaces as reference. **Note** • You can select multiple surfaces only if the datum plane is not selected. • Select a reference surface only if open curve-chain loops are selected.
`PRO_E_SHUT_SRF_CLOSE_ALL`	`PRO_VALUE_TYPE_BOOLEAN`	Specifies an option to include all loops while creating shut off surfaces. The valid value for this element is `PRO_SHUT_SRF_SHUT_ALL`.

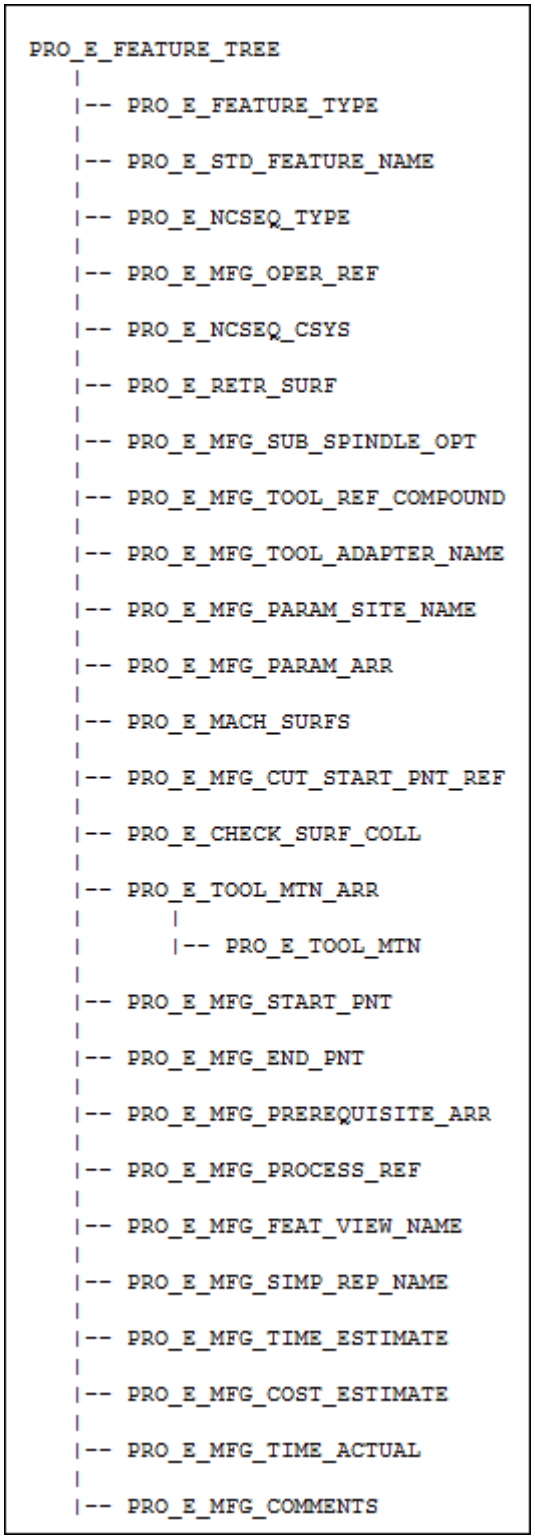
Element Trees: Manufacturing Round and Chamfer

This section describes how to construct and access the element tree for manufacturing a round and chamfer feature. It also describes how to create, redefine, and access the properties of these features.

Manufacturing Round And Chamfer Element Tree

The element tree for the Round and Chamfer feature is documented in the header file `ProMfgFeatRoundChamferMilling.h` and is as shown in the following figure:

Element Tree for Round And Chamfer feature



The following table describes the elements in the element tree for the Round and Chamfer feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Engraving_1.
PRO_E_SEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid values for this element are: • PRO_SEQ_ROUND_MILL— For round sequence. • PRO_SEQ_CHAMFER_MILL— For chamfer sequence.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_SEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the sequee coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676 .
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the

Element ID	Data Type	Description
NAME		name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MACH_SURFS	Compound	Mandatory compound element. Specifies the machining surfaces compound definition. For more information, refer to the section Element Trees: Machining Surfaces on page 1573.
PRO_E_MFG_CUT_START_PNT_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Select a datum point, or a point on the bottom edges of the machining surfaces. **Note** This element is applicable only when the machining surfaces form a closed loop. It allows to start machining at the location nearest to the selected point.
PRO_E_CHECK_SURF_COLL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the check surfaces collection. For more information, refer to the section Checking Surfaces on page 1687.
PRO_E_TOOL_MTN_ARR	Array	Optional element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Optional compound element. Specifies the tool motion compound specifications. The applicable tool motion type for this element are:

Element ID	Data Type	Description
		• PRO_TM_TYPE_ROUND_MILLING— For Round Sequence. For more information, refer to the section Tool Motion — Round Milling on page 1779. • PRO_TM_TYPE_CHAMFER_MILLING— For Chamfer sequence. For more information, refer to the section Tool Motion — Chamfer Milling on page 1775. • PRO_TM_TYPE_FOLLOW_CUT. For more information, refer to the section Tool Motion — Follow Cut on page 1770 . • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more

Element ID	Data Type	Description
		information, refer to the section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_HELICAL_APPROACH. For more information, refer to the section Tool Motion — Helical Approach on page 1717. • PRO_TM_TYPE_HELICAL_EXIT. For more information, refer to the section Tool Motion — Helical Exit on page 1720. • PRO_TM_TYPE_RAMP_APPROACH. For more information, refer to the section Tool Motion — Ramp Approach on page 1758.

Element ID	Data Type	Description
		• PRO_TM_TYPE_RAMP_EXIT. For more information, refer to the section Tool Motion — Ramp Exit on page 1760. PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767. The applicable tool motion types for 4 and 5-axis machining are: • PRO_TM_TYPE_ALONG_AXIS_APPROACH. For more information, refer to the section Approach Along Tool Axis on page 1690 . • PRO_TM_TYPE_ALONG_AXIS_EXIT. For more information, refer to the section Exit Along Tool Axis on page 1692 .
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequees. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the referee selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step.

Element ID	Data Type	Description
		Note This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: Engraving Step

This section describes how to construct and access the element tree for a engraving step. It also describes how to create, redefine, and access the properties of these features.

The Engraving Step Element Tree

The element tree for the Engraving Step feature is documented in the header file `ProMfgFeatEngraving.h` and is as shown in the following figure:

Element Tree for Engraving feature

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_NCSEQ_TYPE
    |
    |-- PRO_E_MFG_OPER_REF
    |
    |-- PRO_E_MFG_SEQ_NUM_AXES_OPT
    |
    |-- PRO_E_NCSEQ_CSYS
    |
    |-- PRO_E_RETR_SURF
    |
    |-- PRO_E_MFG_SUB_SPINDLE_OPT
    |
    |-- PRO_E_MFG_TOOL_REF_COMPOUND
    |
    |-- PRO_E_MFG_TOOL_ADAPTER_NAME
    |
    |-- PRO_E_MFG_PARAM_SITE_NAME
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_MFG_MACH_CRVS
    |       |
    |       |-- PRO_E_STD_CURVE_COLLECTION_APPL
    |       |
    |       |-- PRO_E_MFG_CURVE_FEAT
    |
    |-- PRO_E_MFG_NORM_SRFS
    |       |
    |       |-- PRO_E_STD_SURF_COLLECTION_APPL
    |       |
    |       |-- PRO_E_MFG_NORM_GEOM
    |
    |-- PRO_E_MFG_4_AXIS_PLANE
    |
    |-- PRO_E_TOOL_MTN_ARR
    |       |
    |       |-- PRO_E_TOOL_MTN
    |
    |-- PRO_E_MFG_START_PNT
    |
    |-- PRO_E_MFG_END_PNT
    |
    |-- PRO_E_MFG_PREREQUISITE_ARR
    |
    |-- PRO_E_MFG_PROCESS_REF
    |
    |-- PRO_E_MFG_FEAT_VIEW_NAME
    |
    |-- PRO_E_MFG_SIMP_REP_NAME
    |
    |-- PRO_E_MFG_TIME_ESTIMATE
    |
    |-- PRO_E_MFG_COST_ESTIMATE
    |
    |-- PRO_E_MFG_TIME_ACTUAL
    |
    |-- PRO_E_MFG_COMMENTS
```

The following table describes the elements in the element tree for the Engraving feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence. The default value is Engraving_1.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is PRO_NCSEQ_GROOVE_MILL.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_MFG_SEQ_NUM_AXES_OPT	PRO_VALUE_TYPE_INT	Specifies the number of controlled axes. The valid values for this element are: • 3— Default value. • 4— Used for machining with tool axis parallel to the plane specified in PRO_E_MFG_4_AXIS_PLANE. This is applicable if work center allows 4-axis, or 5-axis machining. • 5— Used for 5-axis machining. This is applicable if work center allows 5-axis machining.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673 .
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page

Element ID	Data Type	Description
		1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676 .
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677 .
PRO_E_MFG_MACH_CRVS	Compound	Mandatory compound element. Specifies the machining curves compound definition.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Specifies the curve collection. **Note** This element is mandatory if the element PRO_E_MFG_CURVE_FEAT is not set.
PRO_E_MFG_CURVE_FEAT	PRO_VALUE_TYPE_SELECTION	Specifies the curve feature selection. This element supports multiple selections. **Note** This element is mandatory if the element PRO_E_STD_CURVE_COLLECTION_APPL is not set.
PRO_E_MFG_NORM_SRFS	Compound	Optional element. Specifies the normal surfaces compound definition.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Specifies the normal surfaces collection and is used for tool axis

Element ID	Data Type	Description
		control. **Note** • This element is mandatory if the element PRO_E_STD_CURVE_COLLECTION_APPL is set with curves. • This element is ignored for 3-axis machining.
PRO_E_MFG_NORM_GEOM	PRO_VALUE_TYPE_SELECTION	Specifies the normal surface selection and is used for tool axis control. **Note** • This element is mandatory if the element PRO_E_STD_CURVE_COLLECTION_APPL is set with edges. • This element is ignored for 3-axis machining.
PRO_E_MFG_4_AXIS_PLANE	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum plane or planar surface. **Note** • This element is mandatory for 4-axis machining, when the element PRO_E_MFG_SEQ_NUM_AXES_OPT is set to 4. • This element is ignored for 3-axis and 5-axis machining.
PRO_E_TOOL_MTN_ARR	Array	Optional element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Mandatory compound element. Specifies the tool motion compound specifications. The applicable tool motion type for this element are: • PRO_TM_TYPE_GROOVE_MILLING. For more information, refer to the section Tool Motion — Groove Milling on page 1778. • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool

Element ID	Data Type	Description
		Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN.

Element ID	Data Type	Description
		For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_HELICAL_APPROACH. For more information, refer to the section Tool Motion — Helical Approach on page 1717. • PRO_TM_TYPE_HELICAL_EXIT. For more information, refer to the section Tool Motion — Helical Exit on page 1720. • PRO_TM_TYPE_RAMP_APPROACH. For more information, refer to the section Tool Motion — Ramp Approach on page 1758. • PRO_TM_TYPE_RAMP_EXIT. For more information, refer to the section Tool Motion — Ramp Exit on page 1760. PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767. The applicable tool motion types for 4 and 5-axis machining are: • PRO_TM_TYPE_ALONG_AXIS_APPROACH. For more information, refer to the section Approach Along Tool Axis on page 1690. • PRO_TM_TYPE_ALONG_AXIS_EXIT. For more information, refer to the section Exit Along Tool Axis on page 1692.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified

Element ID	Data Type	Description
		position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_ NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step.

Element ID	Data Type	Description
		Note This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

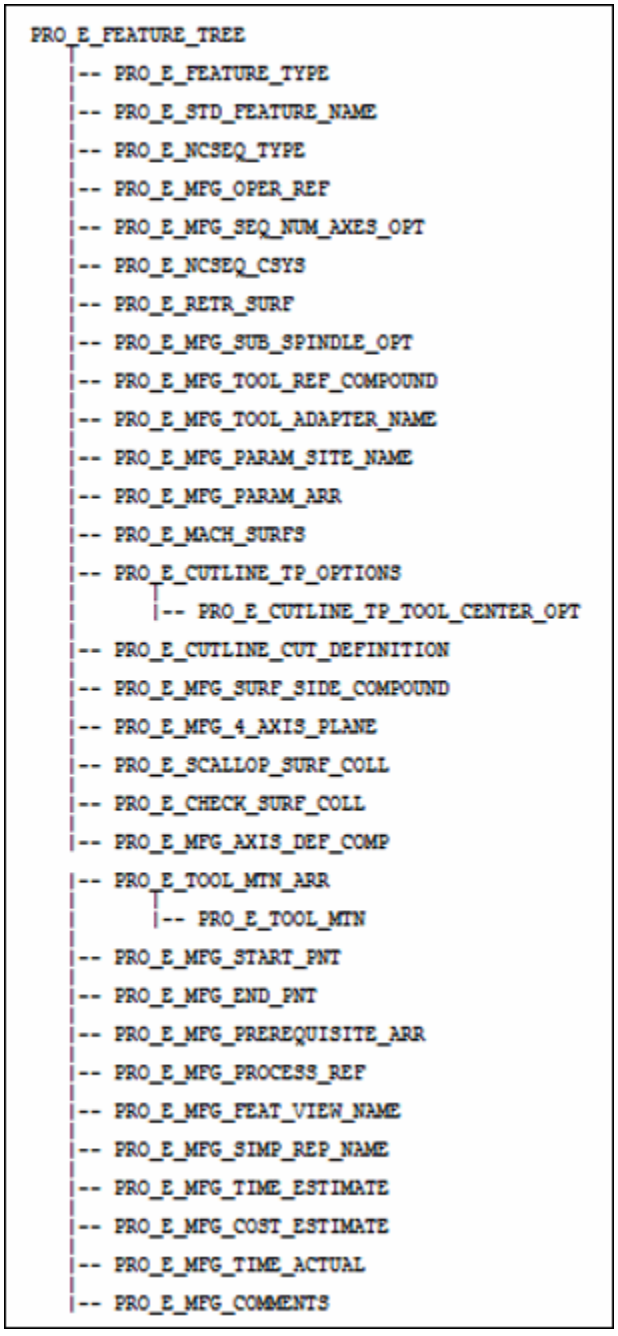
Element Trees: Manufacturing Cutline Milling Sequence

This section describes how to construct and access the element tree for a cutline milling sequence. It also describes how to create, redefine, and access the properties of these features.

The Cutline Milling Element Tree

The element tree for the cutline milling sequence is documented in the header file `ProMfgFeatCutlineMilling.h`, and is as shown in the following figure:

Element Tree for Cutline Milling feature



Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the Creo NC sequence.

Element ID	Data Type	Description
		The default value is `Cutline_Milling_1`.
`PRO_E_NCSEQ_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies the type of Creo NC sequence. The valid value for this element is `PRO_NCSEQ_CUTLINE_MILL`.
`PRO_E_MFG_OPER_REF`	`PRO_VALUE_TYPE_SELECTION`	Mandatory element. Specifies the operation feature selection.
`PRO_E_MFG_SEQ_NUM_AXES_OPT`	`PRO_VALUE_TYPE_INT`	Specifies the number of controlled axes. The valid values for this element are: • `3`—Default value. • `4`—Used for machining with tool axis parallel to the plane specified in `PRO_E_MFG_4_AXIS_PLANE`. This is applicable if work center allows 4-axis, or 5-axis machining. • `5`—Used for 5-axis machining. This is applicable if work center allows 5-axis machining.
`PRO_E_NCSEQ_CSYS`	`PRO_VALUE_TYPE_SELECTION`	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
`PRO_E_RETR_SURF`	`Compound`	Mandatory compound element. Specifies retract compound definition. For more information, refer to the section Retract Elements on page 1673.
`PRO_E_MFG_SUB_SPINDLE_OPT`	`PRO_VALUE_TYPE_INT`	Optional Element. Specifies the type of spindle assigned to the sequence. This element can be used when two parts are machined during the same operation in different spindles, that is in the main spindle and in the sub spindle. The valid values for this element are defined by the enumerated type `ProSubSpindleOpt`. For more information on the values of `ProSubSpindleOpt`, refer to the section Spindle Types on page 1690
`PRO_E_MFG_TOOL_REF_COMPOUND`	`Compound`	Mandatory compound element. Specifies tool reference compound definition. For more information, refer to the section Tool Reference on page 1676.

Element ID	Data Type	Description
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MACH_SURFS	Compound	Mandatory compound element. Specifies the machining surfaces compound definition. For more information, refer to the section Element Trees: Machining Surfaces on page 1573.
PRO_E_CUTLINE_TP_OPTIONS	Compound	Optional element. Specifies tool path options.
PRO_E_CUTLINE_TP_TOOL_CENTER_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the tool center curve option.
PRO_E_CUTLINE_CUT_DEFINITION	Compound	Mandatory element. Specifies the cutline array compound definition. For more information on the sub elements, For more information, refer to the section Element Tree for PRO_E_CUTLINE_CUT_DEFINITION on page 1642.
PRO_E_MFG_SURF_SIDE_COMPOUND	Compound	Optional element. Specifies the surface side definition. For more information, refer to the section Manufacturing Surface Side on page 1566.
PRO_E_MFG_4_AXIS_PLANE	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum plane or planar surface. **Note** This element is mandatory for 4-axis machining, when the element PRO_E_MFG_SEQ_NUM_AXES_OPT is set to 4 and is ignored for 3-axis and 5-axis machining.

Element ID	Data Type	Description
PRO_E_SCALLOP_SURF_COLL	Compound	Optional element. Specifies the scallop surfaces compound definition.
PRO_E_CHECK_SURF_COLL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the check surfaces collection. For more information, refer to the section Checking Surfaces on page 1687.
PRO_E_MFG_AXIS_DEF_COMP	Compound	Optional element. Specifies the compound element for the axis definition.
PRO_E_TOOL_MTN_ARR	Array	Optional element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	Optional element. Specifies the tool motion compound specifications. The applicable tool motion types for this element are : • PRO_TM_TYPE_CUTLINE_MILLING. For more information, refer to the section The Cutline Milling Element Tree on page 1635. • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the

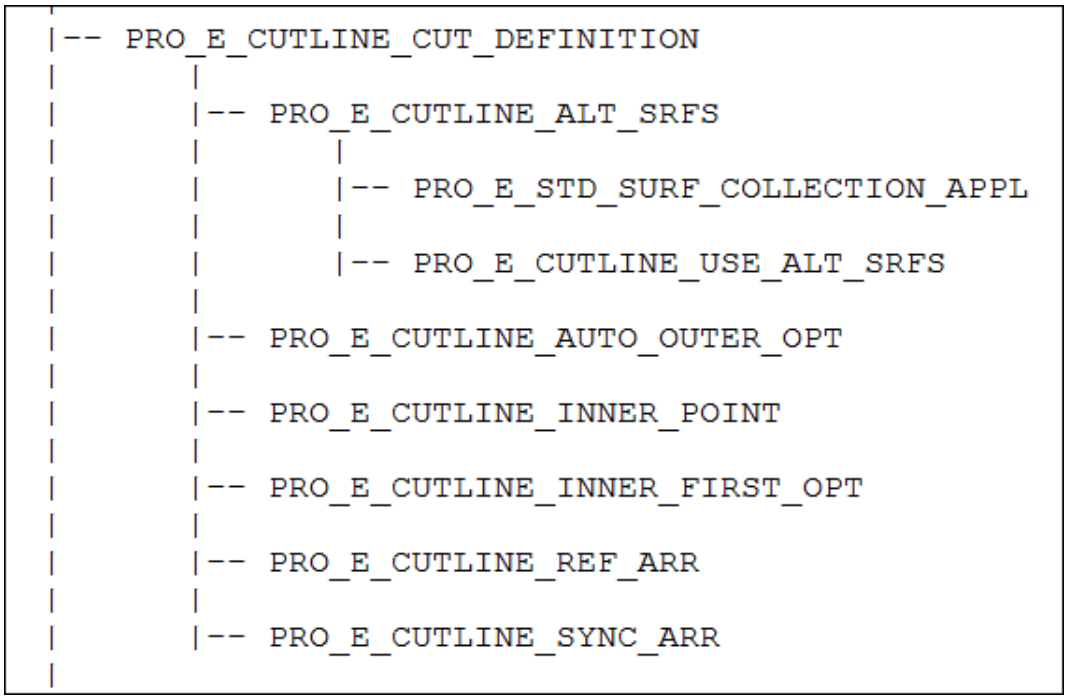
Element ID	Data Type	Description
		section Tool Motion — Tangent Approach on page 1726. • PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_HELICAL_APPROACH. For more information, refer to the section Tool Motion — Helical Approach on page 1717. • PRO_TM_TYPE_HELICAL_EXITFor more information, refer to the section Tool Motion — Helical Exit on page 1720. • PRO_TM_TYPE_RAMP_APPROACH. For more information, refer to the section Tool Motion — Ramp Approach on page 1758. • PRO_TM_TYPE_RAMP_EXIT. For more information, refer to the section Tool Motion — Ramp Exit on page 1760. • PRO_TM_TYPE_CONNECT. For more information, refer to the section Tool Motion — Connect on page 1762.

Element ID	Data Type	Description
		• PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767. The applicable tool motion types for 4 and 5-axis machining are: • PRO_TM_TYPE_ALONG_AXIS_APPROACH. For more information, refer to the section Approach Along Tool Axis on page 1690. • PRO_TM_TYPE_ALONG_AXIS_EXIT. For more information, refer to the section Exit Along Tool Axis on page 1692.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682 .
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane, axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name.

Element ID	Data Type	Description
		This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Tree for PRO_E_CUTLINE_CUT_DEFINITION

The element tree for PRO_E_CUTLINE_CUT_DEFINITION is as shown in the figure below:



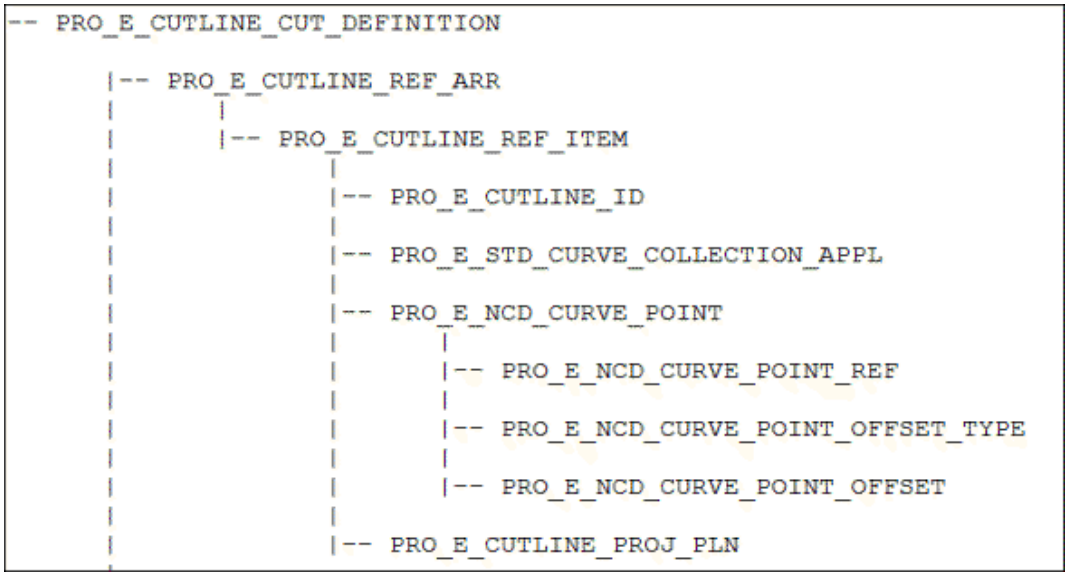
The following table lists the sub elements of the element `PRO_E_CUTLINE_CUT_DEFINITION` defined in the header file `ProMfgFeatCutlineMilling.h`.

Element ID	Data Type	Description
`PRO_E_CUTLINE_ALT_SRFS`	`Compound`	Optional element. Specifies the cutline alternate surface compound definition.
`PRO_E_STD_SURF_COLLECTION_APPL`	`Surface Collection`	Optional element. Specifies the cutline alternate surface collection.
`PRO_E_CUTLINE_USE_ALT_SRFS`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the cutline alternate surface option definition.
`PRO_E_CUTLINE_AUTO_OUTER_OPT`	`PRO_VALUE_TYPE_INT`	Specifies the auto cutline option definition. The valid values for this element are: • `PRO_B_FALSE`—Specifies that cutlines are being defined manually. • `PRO_B_TRUE`—Specifies that cutlines are being defined automatically.
`PRO_E_CUTLINE_INNER_POINT`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Specifies the auto cutline inner point selection.
`PRO_E_CUTLINE_INNER_FIRST_OPT`	`PRO_VALUE_TYPE_INT`	Specifies the auto cutline inner first option definition. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_B_FALSE—Specifies that outer auto cutline is first. • PRO_B_TRUE—Specifies that innner auto cutline is first.
PRO_E_CUTLINE_REF_ARR	Array	Mandatory element. Specifies an array of cutlines.
PRO_E_CUTLINE_SYNC_ARR	Array	Optional element. Specifies an array of cutline synchronization lines.

Element Tree for PRO_E_CUTLINE_REF_ARR

The element tree for PRO_E_CUTLINE_REF_ARR is as shown in the figure below:

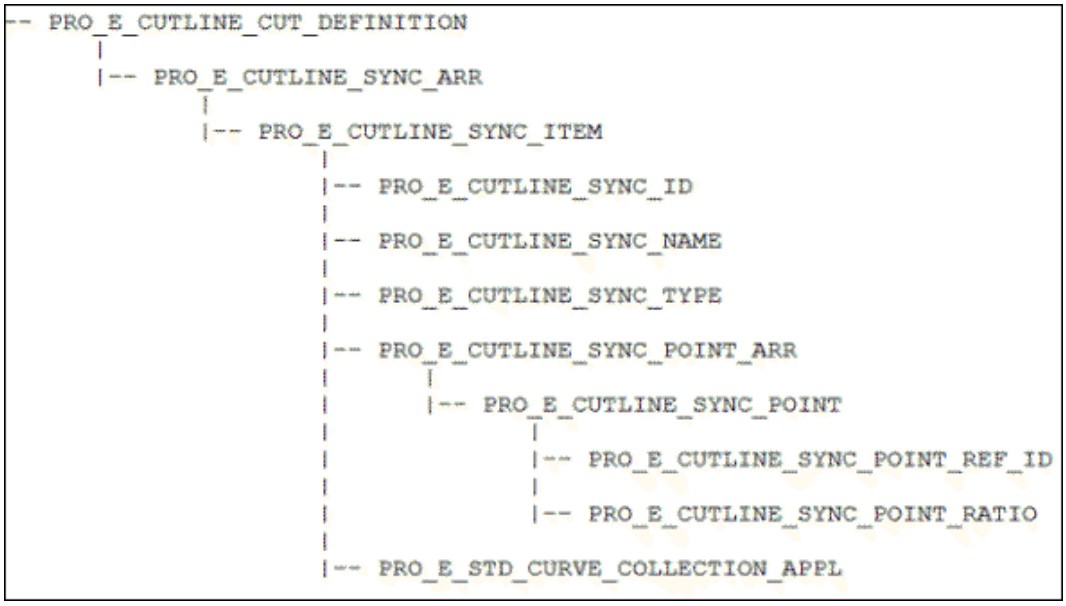


The following table lists the sub elements of the element PRO_E_CUTLINE_REF_ARR defined in the header file ProMfgFeatCutlineMilling.h.

Element ID	Data Type	Description
PRO_E_CUTLINE_REF_ITEM	Compound	Specifies the cutline compound definition. **Note** This element is mandatory if the element PRO_E_CUTLINE_AUTO_OUTER_OPT is set to PRO_B_FALSE and is not used if PRO_E_CUTLINE_AUTO_OUTER_OPT is set to PRO_B_TRUE.
PRO_E_CUTLINE_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the cutline id.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Mandatory element. Specifies the cutline chain collection.
PRO_E_NCD_CURVE_POINT	Compound	Optional element. Specifies the cutline start point compound definition.
PRO_E_NCD_CURVE_POINT_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the cutline start point reference.
PRO_E_NCD_CURVE_POINT_OFFSET_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the cutline start point offset type.
PRO_E_NCD_CURVE_POINT_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cutline start point offset definition.
PRO_E_CUTLINE_PROJ_PLN	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the cutline project plane definition.

Element Tree for PRO_E_CUTLINE_SYNC_ARR

The element tree for PRO_E_CUTLINE_SYNC_ARR is as shown in the figure below:

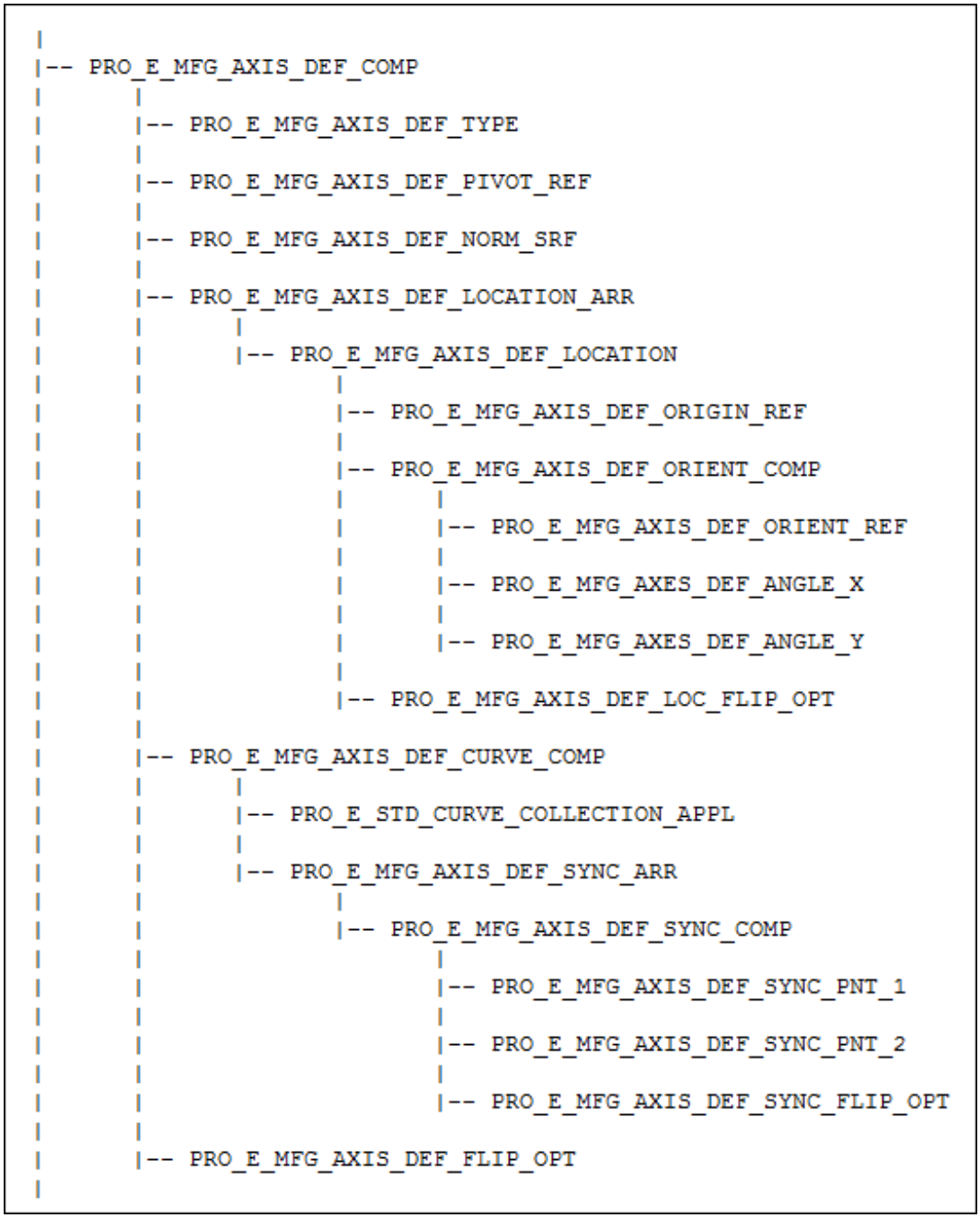


The following table lists the sub elements of the element `PRO_E_CUTLINE_SYNC_ARR` defined in the header file `ProMfgFeatCutlineMilling.h`.

Element ID	Data Type	Description
`PRO_E_CUTLINE_SYNC_ITEM`	`Compound`	Optional element. Specifies the cutline synch line compound definition.
`PRO_E_CUTLINE_SYNC_ID`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the cutline synch line id.
`PRO_E_CUTLINE_SYNC_NAME`	`PRO_VALUE_TYPE_WSTRING`	Optional element. Specifies the cutline synch line name.
`PRO_E_CUTLINE_SYNC_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the cutline synch line type. The valid values for this element are: • `PRO_E_CUTLINE_SYNC_TYPE_POINTS`— Specifies the synch between points on the cutlines. • `PRO_E_CUTLINE_SYNC_TYPE_REF`—Specifies the synch on the reference chains.
`PRO_E_CUTLINE_SYNC_POINT_ARR`	`Array`	Optional element. Specifies an array of cutline synch points.
`PRO_E_CUTLINE_SYNC_POINT`	`Compound`	Optional element. Specifies the cutline synch point compound definition.
`PRO_E_CUTLINE_SYNC_POINT_REF_ID`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the cutline synch point reference ID.
`PRO_E_CUTLINE_SYNC_POINT_RATIO`	`PRO_VALUE_TYPE_DOUBLE`	Optional element. Specifies the cutline synch point ratio.
`PRO_E_STD_CURVE_COLLECTION_APPL`	`Chain Collection`	Optional element. Specifies the cutline synch line chain collection.

Element Tree for PRO_E_MFG_AXIS_DEF_COMP

The element tree for `PRO_E_MFG_AXIS_DEF_COMP` is as shown in the figure below:



PRO_E_MFG_AXIS_DEF_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the axis definition. The valid values for this element are defined in the enumerated type ProAxisDefType and are as follows: • PRO_AXIS_DEF_TYPE_UNDEF • PRO_AXIS_DEF_BY_PIVOT_REF • PRO_AXIS_DEF_BY_LOCATIONS • PRO_AXIS_DEF_BY_TWO_CONTOURS • PRO_AXIS_DEF_BY_NORM_SURF
PRO_E_MFG_AXIS_DEF_PIVOT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the single reference. You can select either a point or an axis. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_PIVOT_REF This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_NORM_SRF	Compound	Normal Surface compound element. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_ NORM_SURF. This element is ignored in all other cases.

PRO_E_MFG_AXIS_DEF_LOCATION_ARR	Array	Specifies an array of locations. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_LOCATIONS. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_LOCATION	Compound	Mandatory element. Specifies the compound element for the location axis definition.
PRO_E_MFG_AXIS_DEF_ORIGIN_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the single reference. You can either select a point on a curve or an edge.
PRO_E_MFG_AXIS_DEF_ORIENT_COMP	Compound	Mandatory element. Specifies the orientation compound element.
PRO_E_MFG_AXIS_DEF_ORIENT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the single reference selection. You can select either a point or an axis. **Note** This element is mandatory, if the elements PRO_E_MFG_AXES_DEF_ANGLE_X and PRO_E_MFG_AXES_DEF_ANGLE_Y are not defined.
PRO_E_MFG_AXES_DEF_ANGLE_X	PRO_VALUE_TYPE_DOUBLE	Specifies the lead angle. The valid range for this element is from —90 to +90. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_ORIENT_REF is not defined.
PRO_E_MFG_AXES_DEF_ANGLE_Y	PRO_VALUE_TYPE_DOUBLE	Specifies the tilt angle. The valid range for this element is from —90 to +90. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_ORIENT_REF is not defined.

PRO_E_MFG_AXIS_DEF_LOC_FLIP_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the flip direction at a location. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction will be reversed. • PRO_B_FALSE—Specifies that the direction will remain the same.
PRO_E_MFG_AXIS_DEF_CURVE_COMP	Compound	Specifies the compound element for the pivot curve. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_TWO_CONTOURS. This element is ignored in all other cases.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Mandatory element. Specifies a general compound element for chain collection.
PRO_E_MFG_AXIS_DEF_SYNC_ARR	Array	Optional element. Specifies the synchronization array.
PRO_E_MFG_AXIS_DEF_SYNC_COMP	Compound	Optional element. Specifies the synchronization compound element.
PRO_E_MFG_AXIS_DEF_SYNC_PNT_1	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the single reference selection. Select a point on the trajectory curve.
PRO_E_MFG_AXIS_DEF_SYNC_PNT_2	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the single reference selection. Select a point on the pivot curve.
PRO_E_MFG_AXIS_DEF_SYNC_FLIP_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the flip direction.The valid values for this element are: • PRO_B_TRUE—Specifies that the tool motion is flipped in the reverse direction. • PRO_B_FALSE—Specifies that the direction will remain the same.
PRO_E_MFG_AXIS_DEF_FLIP_OPT	PRO_VALUE_TYPE_INT	Specifies the flip direction. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction will be reversed. • PRO_B_FALSE—Specifies that the direction will remain the same.

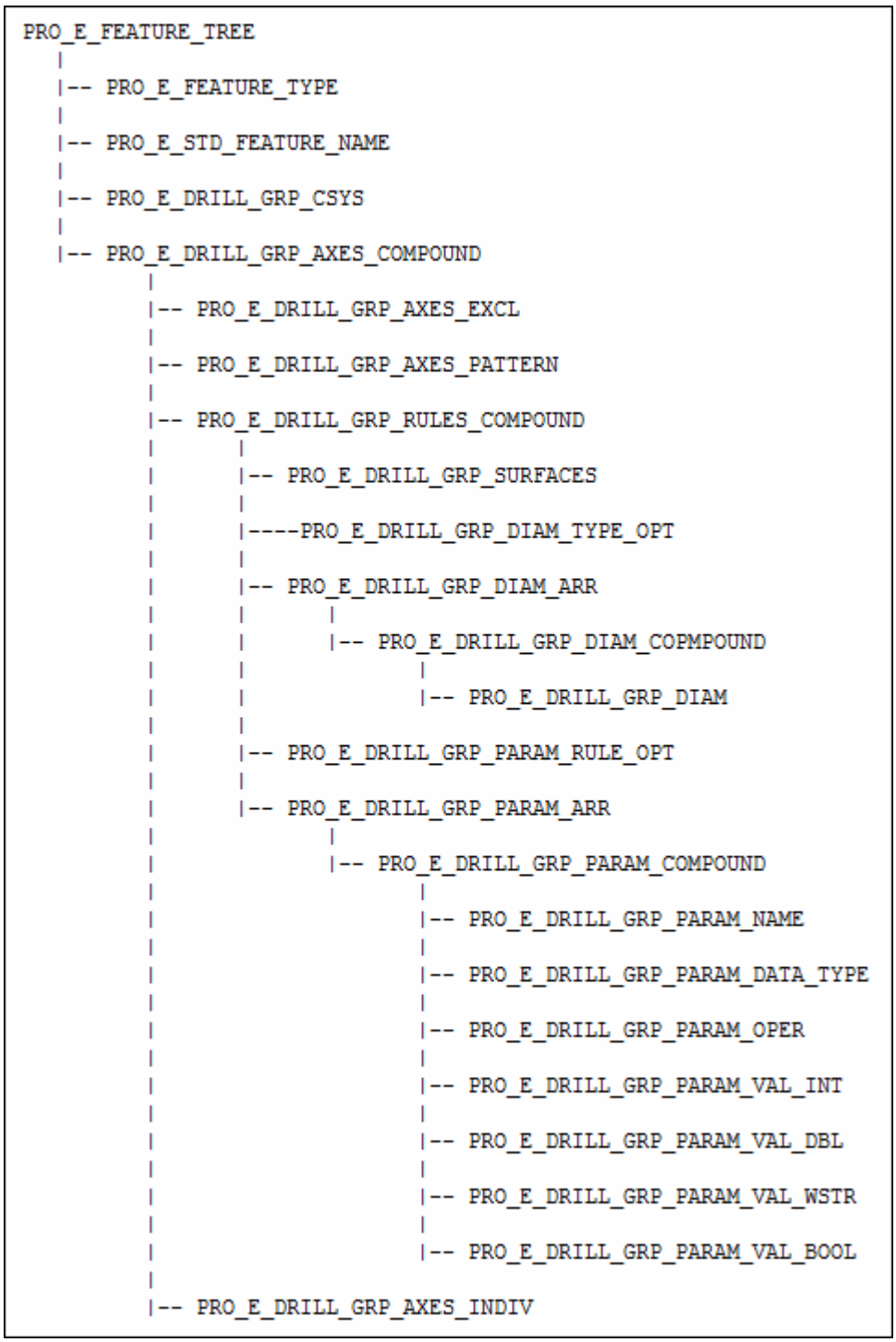
Element Trees: Manufacturing Drill Group Feature

This section describes how to construct and access the element tree for a Drill Group feature. It also describes how to create, redefine, and access the properties of these features.

The Drill Group Feature Element Tree

The element tree for the drill group feature is documented in the header file `ProMfgFeatDrillGroup.h`, and is as shown in the following figure:

Element Tree for Drill Group feature



The following table describes the elements in the element tree for the Drill Group feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_OPERATION.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name for the NC feature. The default value is DRILL_GROUP_1.
PRO_E_DRILL_GRP_CSYS	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum coordinate system. **Note** This element is not supported for the Creo Parametric 2.0 release.
PRO_E_DRILL_GRP_AXES_COMPOUND	Compound	Mandatory element. This compound element specifies information about location of holes to be drilled. **Note** You can use this element only when at least one of the following child elements are defined: • PRO_E_DRILL_GRP_AXES_INDIV • PRO_E_DRILL_GRP_AXES_PATTERN • PRO_E_DRILL_GRP_SURFACES • PRO_E_DRILL_GRP_DIAM_ARR • PRO_E_DRILL_GRP_PARAM_ARR
PRO_E_DRILL_GRP_AXES_EXCL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of axes of holes to be excluded for machining. This element supports multiple selections.
PRO_E_DRILL_GRP_AXES_PATTERN	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of axes of patterned holes. This element supports multiple selections.

Element ID	Data Type	Description
		Note If a pattern leader is selected, all the holes in the pattern will be collected.
PRO_E_DRILL_GRP_RULES_COMPOUND	Compound	Optional element. This compound element specifies the information about hole search rules.
PRO_E_DRILL_GRP_SURFACES	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of surfaces or quilts with holes. This element supports multiple selections.
PRO_E_DRILL_GRP_DIAM_TYPE_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of hole diameters that can be collected in the element PRO_E_DRILL_GRP_DIAM_ARR. The type of hole diameter is specified using the enumerated data type ProHolesetDiamType. The valid values are: • PRO_HSET_ALL_DIAMS—This is the default value. Specifies that diameters of both solid surfaces and cosmetic threads can be collected. **Note** If the element PRO_E_DRILL_GRP_DIAM_TYPE_OPT is not defined, then by default, the hole diameter of type PRO_HSET_ALL_DIAMS is used. • PRO_HSET_HOLE_DIAMS—Specifies that diameters only of solid surfaces can be collected. • PRO_HSET_THREAD_DIAMS—Specifies that diameters only of cosmetic threads can be collected.
PRO_E_DRILL_GRP_DIAM_ARR	Array	Optional element. Specifies an array of diameters of holes to be machined.
PRO_E_DRILL_GRP_DIAM_COPMPOUND	Compound	Optional element. Specifies a compound definition of a hole diameter.
PRO_E_DRILL_GRP_DIAM	PRO_VALUE_TYPE_DOUBLE	Specifies the diameter of the hole

Element ID	Data Type	Description
		to be machined. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_DIAM_COPMPOUND element.
PRO_E_DRILL_GRP_PARAM_RULE_OPT	PRO_VALUE_TYPE_INT	Specifies the type of query that must be used to search for holes to machine. The query type is specified using the enumerated data type ProHsetParamRuleOpt. The valid values are: • PRO_HSET_BOOL_OPER_OR—Collects holes that satisfy at least one of the search conditions set for a parameter. PRO_HSET_BOOL_OPER_AND—Collects holes that satisfy all the search conditions set for a parameter. The search conditions and parameters are defined in the elements PRO_E_DRILL_GRP_PARAM*.
PRO_E_DRILL_GRP_PARAM_ARR	Array	Optional element. Specifies an array of search conditions to collect the holes for machining.
PRO_E_DRILL_GRP_PARAM_COMPOUND	Compound	Optional element. Specifies a compound definition of a condition to match the hole features with the user defined parameters. **Note** In Creo Parametric, each condition defines expression with user defined parameter name on the left side of the expression and value on the right side for comparison.
PRO_E_DRILL_GRP_PARAM_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the user

Element ID	Data Type	Description
		defined parameter. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element.
PRO_E_DRILL_GRP_PARAM_DATA_TYPE	PRO_VALUE_TYPE_INT	Specifies the data type of specified value using the enumerated type ProParamvalueType. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element.
PRO_E_DRILL_GRP_PARAM_OPER	PRO_VALUE_TYPE_INT	Specifies the type of the expression operator using the enumerated type ProDrillParamOper. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element.
PRO_E_DRILL_GRP_PARAM_VAL_INT	PRO_VALUE_TYPE_INT	Specifies the value of type double. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element for double data type and is ignored for other data types.
PRO_E_DRILL_GRP_PARAM_VAL_DBL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of type integer. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element for integer data type and is ignored for other data types.

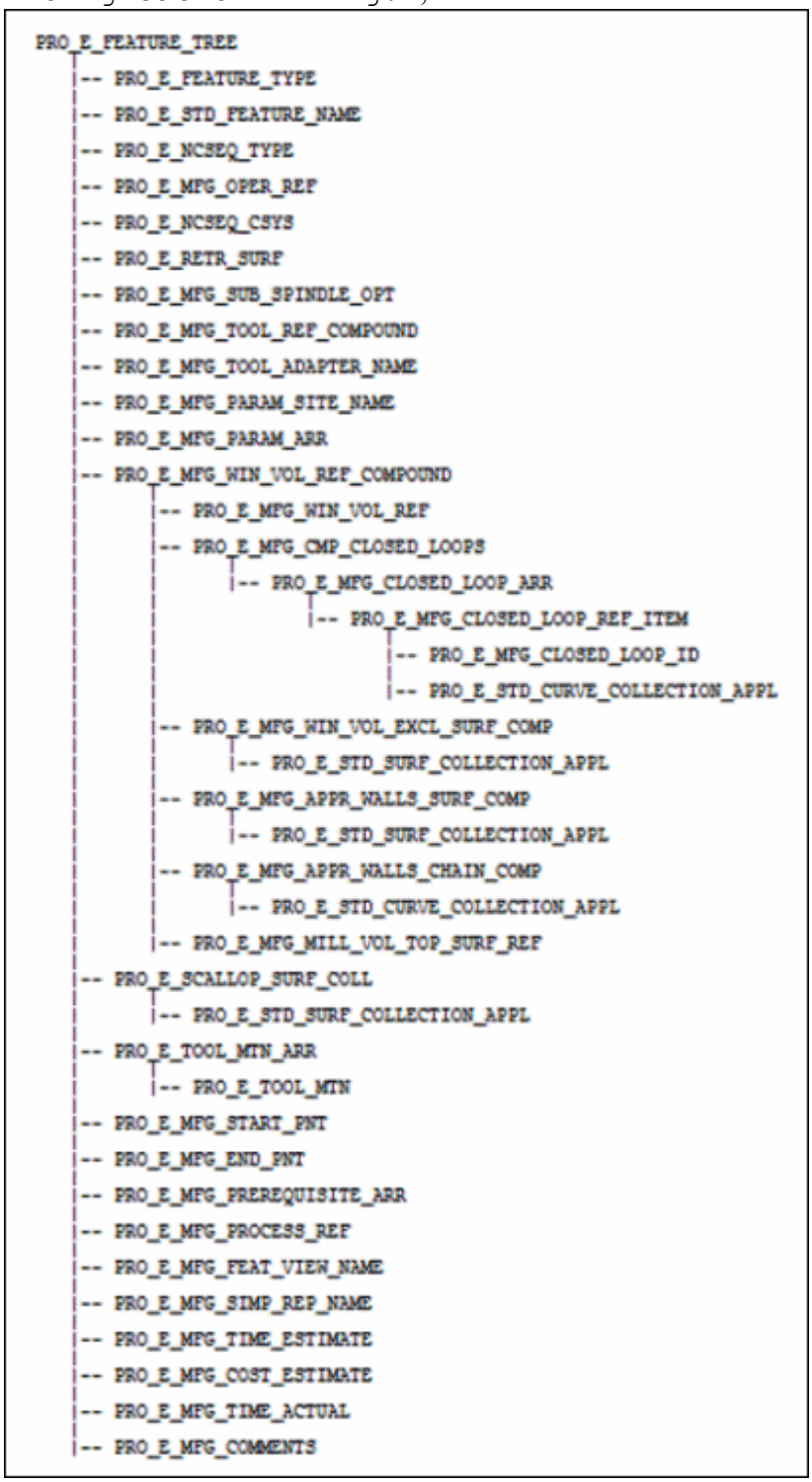
Element ID	Data Type	Description
PRO_E_DRILL_GRP_PARAM_VAL_WSTR	PRO_VALUE_TYPE_WSTRING	Specifies the value of type string. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element for string data type and is ignored for other data types.
PRO_E_DRILL_GRP_PARAM_VAL_BOOL	PRO_VALUE_TYPE_INT	Specifies the value of type ProBoolean. **Note** This element is a mandatory child of PRO_E_DRILL_GRP_PARAM_COMPOUND element for ProBoolean data type and is ignored for other data types.
PRO_E_DRILL_GRP_AXES_INDIV	PRO_VALUE_TYPE_SELECTION	Specifies the selection of datum axes. This element supports multiple selections. **Note** This element is mandatory if none of the following elements are defined: • PRO_E_DRILL_GRP_AXES_PATTERN • PRO_E_DRILL_GRP_SURFACES • PRO_E_DRILL_GRP_DIAM_ARR • PRO_E_DRILL_GRP_PARAM_ARR

Manufacturing Volume Milling Feature

This section describes how to construct and access the element tree for a Volume Milling feature. It also describes how to create, redefine, and access the properties of this feature.

The Volume Milling Feature Element Tree

The element tree for the Volume Milling feature is documented in the header file `ProMfgFeatVolMilling.h`, and is as shown in the following figure:



Element Tree for Volume Milling feature

The following table describes the elements in the element tree for the Volume Milling feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the feature. The valid value for this element is PRO_FEAT_MILL.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the default name for the feature.
PRO_E_NCSEQ_TYPE	PRO_VALUE_TYPE_INT	Specifies the feature form and should be of type PRO_NCSEQ_VOL_MILL only.
PRO_E_MFG_OPER_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the operation feature selection.
PRO_E_NCSEQ_CSYS	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum coordinate system that will be used as the coordinate system for the Creo NC sequence.
PRO_E_RETR_SURF	Compound	Mandatory compound element. Specifies retract definition. For more information, refer to the section Retract Elements on page 1673.
PRO_E_MFG_SUB_SPINDLE_OPT	PRO_VALUE_TYPE_INT	Optional Element. Specifies the type of spindle assigned to the sequence. The default value for this element is PRO_MFG_MAIN_SPINDLE.The valid values for this element are defined by the enumerated type ProSubSpindleOpt. For more information on the values of ProSubSpindleOpt, refer to the section Spindle Types on page 1690
PRO_E_MFG_TOOL_REF_COMPOUND	Compound	Mandatory compound element. Specifies tool reference definition. For more information, refer to the section Tool Reference on page 1676.
PRO_E_MFG_TOOL_ADAPTER_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tool adapter model name.
PRO_E_MFG_PARAM_SITE_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the site file with default values for manufacturing parameters. **Note** The site file name will be ignored if the site does not exist in the manufacturing model.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677.
PRO_E_MFG_WIN_VOL_REF_COMPOUND	Compound	Mandatory element. Specifies the machining reference compound specification.
PRO_E_MFG_WIN_VOL_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the selection of mill window or mill volume feature.
PRO_E_MFG_CMP_CLOSED_LOOPS	Compound	Optional element. Specifies the closed loop compound specification.
PRO_E_MFG_CLOSED_LOOP_ARR	Array	Optional element. Specifies an array of closed loop specifications.
PRO_E_MFG_CLOSED_LOOP_REF_ITEM	Compound	Optional element. Specifies the closed loop specification.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Optional element. Specifies an excluded surfaces compound specification for chain collection.
PRO_E_MFG_WIN_VOL_EXCL_SURF_COMP	Compound	
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Optional element. Specifies surface collection for the creation of the volume milling sequence.
PRO_E_MFG_APPR_WALLS_SURF_COMP	Compound	Optional element. Specifies the approach walls surfaces compound specification. Use this element only if mill volume is selected as a machining reference.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Optional element. Specifies surface collection for the creation of the volume milling sequence.
PRO_E_MFG_APPR_WALLS_CHAIN_COMP	Compound	Optional element. Specifies the approach walls chain compound specification. Use this element only if mill volume is selected as a machining reference.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Optional element. Specifies the chain collection for the creation of the volume milling sequence.
PRO_E_MFG_MILL_VOL_TOP_SURF_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of the top surface. Use this element only if mill volume is selected as a machining reference.
PRO_E_SCALLOP_SURF_COLL	Compound	Optional element. Specifies the scallop surfaces compound definition.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Optional element. Specifies the collection of the scallop surfaces.

Element ID	Data Type	Description
PRO_E_CHECK_SURF_COLL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the check surfaces compound definition. The element tree for the Checking Surfaces is defined in the header file ProMfgElemCheckSurf.h. For more information, refer to the section Checking Surfaces on page 1687 for more information on the element tree.
PRO_E_TOOL_MTN_ARR	Array	Mandatory element. Specifies an array of tool motions.
PRO_E_TOOL_MTN	Compound	• PRO_TM_TYPE_VOLUME_MILLING. For more information, refer to the section Tool Motion — Volume Mill Cut on page 1781. • PRO_TM_TYPE_FOLLOW_CUT. For more information, refer to the section Tool Motion — Follow Cut on page 1770. • PRO_TM_TYPE_FOLLOW_CURVE. For more information, refer to the section Tool Motion — Follow Curve on page 1694. • PRO_TM_TYPE_GOTO_POINT. For more information, refer to the section Tool Motion — Go To Point on page 1696. • PRO_TM_TYPE_GO_DELTA. For more information, refer to the section Tool Motion — Go Delta on page 1700. • PRO_TM_TYPE_GOHOME. For more information, refer to the section Tool Motion — Go Home on page 1704. • PRO_TM_TYPE_PLUNGE. For more information, refer to the section Tool Motion — Plunge on page 1772. • PRO_TM_TYPE_GO_RETRACT. For more information, refer to the section Tool Motion — Go Retract on page 1708. • PRO_TM_TYPE_TANGENT_APPROACH. For more information, refer to the section Tool Motion — Tangent Approach on page 1726.

Element ID	Data Type	Description
		• PRO_TM_TYPE_TANGENT_EXIT. For more information, refer to the section Tool Motion — Tangent Exit on page 1728. • PRO_TM_TYPE_NORMAL_APPROACH. For more information, refer to the section Tool Motion — Normal Approach on page 1710. • PRO_TM_TYPE_NORMAL_EXIT. For more information, refer to the section Tool Motion — Normal Exit on page 1713. • PRO_TM_TYPE_LEAD_IN. For more information, refer to the section Tool Motion — Lead In on page 1706. • PRO_TM_TYPE_LEAD_OUT. For more information, refer to the section Tool Motion — Lead Out on page 1715. • PRO_TM_TYPE_HELICAL_APPROACH. For more information, refer to the section Tool Motion — Helical Approach on page 1717. • PRO_TM_TYPE_HELICAL_EXITFor more information, refer to the section Tool Motion — Helical Exit on page 1720. • PRO_TM_TYPE_CL_COMMAND. For more information, refer to the section Tool Motion — CL Command on page 1767.
PRO_E_MFG_START_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to start the machining at the specified position.
PRO_E_MFG_END_PNT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the datum point selection. It allows to end the machining at the specified position.
PRO_E_MFG_PREREQUISITE_ARR	Array	Optional element. This array specifies the Ids of the prerequisite sequences. For more information, refer to the section Sequence Prerequisites on page 1682.
PRO_E_MFG_PROCESS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the reference selections such as part, feature, curve, surface, datum plane,

Element ID	Data Type	Description
		axis, point, and datum coordinate. It allows you to create additional geometric references to be used only in special process application. This element supports multiple selections.
PRO_E_MFG_FEAT_VIEW_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the name of the view. This element allows you to associate a specific view with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_SIMP_REP_NAME	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the simplified representation name. This element allows you to associate the specific simplified representation with the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the time estimated for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the cost estimate for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_TIME_ACTUAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the actual time for the machining step. **Note** This element is used only in special process application.
PRO_E_MFG_COMMENTS	PRO_VALUE_TYPE_WSTRING	Specifies the sequence comments.

Element Trees: Skirt Feature

This section describes how to construct and access the element tree for a Skirt Surface Extension feature. It also describes how to create, redefine, and access the properties of these features.

Skirt Surface Extension Feature Element Tree

The element tree for the skirt surface extension feature is documented in the header file `ProMoldSkirtExt.h`, and is as shown in the following figure:

Element Tree for Skirt Surface Extension feature

```
PRO_E_FEATURE_TREE
   |
   |-- PRO_E_FEATURE_TYPE
   |
   |-- PRO_E_FEATURE_FORM
   |
   |-- PRO_E_STD_FEATURE_NAME
   |
   |-- PRO_E_SKIRT_REF_MODEL
   |
   |-- PRO_E_SKIRT_BOUNDARY_REF
   |
   |-- PRO_E_SKIRT_PULL_DIR_COMPOUND
   |       |
   |       |-- PRO_E_DIRECTION_COMPOUND
   |
   |-- PRO_E_SKIRT_EXT_SET_ARR
   |       |
   |       |-- PRO_E_SKIRT_EXT_SET_COMPOUND
   |               |
   |               |-- PRO_E_SKIRT_EXT_SET_REF_IDX
   |               |
   |               |-- PRO_E_SKIRT_EXT_SET_TYPE
   |               |
   |               |-- PRO_E_SKIRT_EXT_SET_CURVE_COMP
   |               |       |
   |               |       |-- PRO_E_STD_CURVE_COLLECTION_APPL
   |               |
   |               |-- PRO_E_SKIRT_EXT_SET_DIR_COMPOUND
   |               |       |
   |               |       |-- PRO_E_DIRECTION_COMPOUND
   |               |
   |               |-- PRO_E_SKIRT_EXT_SET_NEXT_DIR_OPT
   |
   |-- PRO_E_SKIRT_SHUTOFF_EXT_COMPOUND
   |       |
   |       |-- PRO_E_SKIRT_SHUTOFF_EXT_TYPE
   |       |
   |       |-- PRO_E_SKIRT_SHUTOFF_EXT_DIST
   |       |
   |       |-- PRO_E_SKIRT_SHUTOFF_CURVE_COMP
   |       |       |
   |       |       |-- PRO_E_STD_CURVE_COLLECTION_APPL
   |       |
   |       |-- PRO_E_SKIRT_DRAFT_ANGLE
   |       |
   |       |-- PRO_E_SKIRT_SHUTOFF_PLANE_REF
   |
   |-- PRO_E_SKIRT_CREATE_TRANS_OPT
```

The following table describes the elements in the element tree for the Skirt Extension feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the feature. The valid value for this element is PRO_FEAT_DATUM_SURF.
PRO_E_FEATURE_FORM	PRO_VALUE_TYPE_INT	Specifies the feature form and should be of type PRO_SKIRT_EXT only.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the default name for the feature. The default value for this element isSkirt_Extension_1
PRO_E_SKIRT_REF_MODEL	PRO_VALUE_TYPE_SELECTION	Mandatory element. Select the reference model used for creating the parting extension surface. The valid reference for this element is a single PRO_PART.
PRO_E_SKIRT_BOUNDARY_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specify the boundary reference for trimming the outer limits of the skirt surface. The valid reference for this element is either a single PRO_PART or a single PRO_QUILT.
PRO_E_SKIRT_PULL_DIR_COMPOUND	Compound	Specifies the reference for the view direction. The valid reference for this element is PRO_E_DIRECTION_COMPOUND. This element is optional, if the default pull direction exists. The default pull direction is used as a reference for the view direction.
PRO_E_SKIRT_EXT_SET_ARR	Array	Mandatory element. Specifies an array element which contain a set of PRO_E_SKIRT_EXT_SET_COMPOUND elements.
PRO_E_SKIRT_EXT_SET_COMPOUND	Compound	Mandatory element. Specify one element of this type for each compound set PRO_E_SKIRT_EXT_SET_COMPOUND contained in PRO_E_SKIRT_EXT_SET_ARR. Each set provides information about the skirt extension set.
PRO_E_SKIRT_EXT_SET_REF_IDX	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the reference index, which is unique for each compound set PRO_E_SKIRT_EXT_SET_COMPOUND.
PRO_E_SKIRT_EXT_SET_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of extension used for the skirt feature creation. This element defines the direction of the extension. The valid values for this element are defined by the enumerated type ProSkirtExtType and are as

Element ID	Data Type	Description
		follows: • PRO_SKIRT_EXT_TYPE_NORMAL—Specifies that the skirt surface extension will be created normal to the pull direction and will end in a direction normal to the curve. The extension is created between the edge and the boundary reference. • PRO_SKIRT_EXT_TYPE_PARALLEL—Specifies that the skirt surface extension will be created parallel to the pull direction. • PRO_SKIRT_EXT_TYPE_TANGENT—Specifies that the skirt extension is created tangent to the model surface of the selected reference model. • PRO_SKIRT_EXT_TYPE_USER—Specifies that the skirt surface extension is created in the user-defined direction. • PRO_SKIRT_EXT_TYPE_NORMAL_TO_BNDRY—Specifies that the skirt extension is created normal to the pull direction and will end in a direction normal to the boundary.
PRO_E_SKIRT_EXT_SET_CURVE_COMP	Compound	Mandatory element. This element specifies the collection of extension curves.
PRO_E_STD_CURVE_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the curves for the skirt surface extension.
PRO_E_SKIRT_EXT_SET_DIR_COMPOUND	Compound	Specifies the direction reference for building the geometry of the extension. This element is mandatory, only if the element PRO_E_SKIRT_EXT_SET_TYPE is set to the value PRO_SKIRT_EXT_TYPE_USER.
PRO_E_DIRECTION_COMPOUND	Compound	Specifies the direction reference for the extension of surfaces. The valid references for this element are: • Straight Edge • Straight Curve • Planar Surface • Datum Plane

Element ID	Data Type	Description
		• Datum axis • Datum Coordinate System Axis
PRO_E_SKIRT_EXT_SET_NEXT_DIR_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies an option to switch between the tangent or parallel extension solutions. In tangent extensions, this element enables you to switch between the two available tangent extension solutions, whereas in parallel extensions, this element flips the direction of the extension. The valid values for this element are defined by the enumerated type ProSkirtExtNextDirOpt and are as follows: • PRO_SKIRT_EXT_NEXT_DIR_DEFAULT—This is the default value for tangent and parallel extensions. Tangent extensions, use the default base geometry whereas parallel extensions, extend along the view direction. • PRO_SKIRT_EXT_NEXT_DIR_ALTERNATE—In case of tangent extension, the other base geometry is used to create the extension. For parallel extensions, the extension is opposite to the view direction.
PRO_E_SKIRT_SHUTOFF_EXT_COMPOUND	Compound	Mandatory element. This element provides information about the shut-off extension options.
PRO_E_SKIRT_SHUTOFF_EXT_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the extension shut-off used during the skirt feature creation. The valid values for this element are defined by the enumerated type ProSkirtShutoffExtType and are as follows: • PRO_SKIRT_EXT_SHUTOFF_BY_DIST—Defines the shut-off extension by a specified distance. • PRO_SKIRT_EXT_SHUTOFF_BY_BOUND—Defines the shut-off extension till the selected boundary.
PRO_E_SKIRT_SHUTOFF_EXT_DIST	PRO_VALUE_TYPE_DOUBLE	Specifies the shut-off extension distance. This element is

Element ID	Data Type	Description
		mandatory only if the element PRO_E_SKIRT_SHUTOFF_EXT_TYPE is set to the value PRO_SKIRT_EXT_SHUTOFF_BY_DIST.
PRO_E_SKIRT_SHUTOFF_CURVE_COMP	Compound	Specifies the selected boundary for the shut-off extension. This element is mandatory only if the element PRO_E_SKIRT_SHUTOFF_EXT_TYPE is set to the value PRO_SKIRT_EXT_SHUTOFF_BY_BOUND.
PRO_E_SKIRT_DRAFT_ANGLE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the draft angle to define the shut-off extension. The default value for this element is zero degrees.
PRO_E_SKIRT_SHUTOFF_PLANE_REF	PRO_VALUE_TYPE_SELECTION	Specifies a parting plane on which the shut-off surface extension is to be created. It defines how far the drafted surface will be extended. This element is mandatory only if the element PRO_E_SKIRT_SHUTOFF_EXT_TYPE is set to the value PRO_SKIRT_EXT_SHUTOFF_BY_BOUND
PRO_E_SKIRT_CREATE_TRANS_OPT	PRO_VALUE_TYPE_INT	Optional element. Specify a value to create transitions across different sets of skirt extensions.

Skirt Fill Feature

The element tree for the skirt fill feature is documented in the header file ProMoldSkirtFill.h, and is as shown in the following figure:

Element Tree for Skirt Fill feature

```
PRO_E_FEATURE_TREE
    |
    |-- PRO_E_FEATURE_TYPE
    |
    |-- PRO_E_STD_FEATURE_NAME
    |
    |-- PRO_E_SKIRT_REF_MODEL
    |
    |-- PRO_E_SKIRT_PULL_DIR_COMPOUND
    |       |
    |       |-- PRO_E_DIRECTION_COMPOUND
    |
    |-- PRO_E_SKIRT_FILL_SET_ARR
            |
            |-- PRO_E_SKIRT_FILL_SET_COMPOUND
                    |
                    |-- PRO_E_SKIRT_FILL_SET_REF_IDX
                    |
                    |-- PRO_E_SKIRT_FILL_CURVES_COMPOUND
                    |       |
                    |       |-- PRO_E_STD_CURVE_COLLECTION_APPL
                    |
                    |-- PRO_E_SKIRT_FILL_TYPE
                    |
                    |-- PRO_E_SKIRT_FILL_PLANE
                    |
                    |-- PRO_E_SKIRT_FILL_SURF_COMPOUND
                    |       |
                    |       |-- PRO_E_STD_SURF_COLLECTION_APPL
                    |
                    |-- PRO_E_SKIRT_FILL_LOOP_OFFSET
```

The following table describes the elements in the element tree for the Skirt Fill feature.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the feature. The valid value for this element is PRO_FEAT_DATUM_SURF.
PRO_E_FEATURE_FORM	PRO_VALUE_TYPE_INT	Specifies the feature form and should be of type PRO_SKIRT_FILL only.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the default name of the feature. The default value for this element is Skirt_Fill_1
PRO_E_SKIRT_REF_MODEL	PRO_VALUE_TYPE_	Mandatory element. Select the

Element ID	Data Type	Description
	SELECTION	reference model used for creating the skirt surface fill feature. The valid reference for this element is a single PRO_PART
PRO_E_SKIRT_PULL_DIR_COMPOUND	Compound	Specifies the reference for the view direction. The valid reference for this element is PRO_E_DIRECTION_COMPOUND. This element is optional, if the default pull direction exists. The default pull direction is used as a reference for the view direction.
PRO_E_DIRECTION_COMPOUND	Compound	Specifies the direction reference for the skirt surface. The valid references for this element are: • Straight Edge • Straight Curve • Planar Surface • Datum Plane • Datum axis • Datum Coordinate System Axis
PRO_E_SKIRT_FILL_SET_ARR	Array	Mandatory element. Specifies an array element which contains a set of PRO_E_SKIRT_FILL_SET_COMPOUND elements.
PRO_E_SKIRT_FILL_SET_COMPOUND	Compound	Mandatory element. Specify one element of this type for each compound set PRO_E_SKIRT_FILL_SET_COMPOUND contained in PRO_E_SKIRT_FILL_SET_ARR. Each set gives provides information about the skirt fill set.
PRO_E_SKIRT_FILL_SET_REF_IDX	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the reference index, which is a unique value for each compound set PRO_E_SKIRT_FILL_SET_COMPOUND.
PRO_E_SKIRT_FILL_CURVES_COMPOUND	Compound	Mandatory element. Specifies the collection of skirt fill closure curves.
PRO_E_STD_CURVE_COLLECTION_APPL	Compound	Specifies the selection of the curves for the skirt surface fill feature.
PRO_E_SKIRT_FILL_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specify the type of closure to be used for closing inner loops or holes in a skirt parting surface. The valid values for this element are defined by the enumerated type ProSkirtFillType and are as

Element ID	Data Type	Description
		follows: • PRO_SKIRT_FILL_STANDARD—This is the default closure type. • PRO_SKIRT_FILL_MID_PLANE—Specifies the closure from the middle plane. • PRO_SKIRT_FILL_MID_SURF—Specifies the closure from the middle surface. • PRO_SKIRT_FILL_CAP_PLANE—Specifies the closure from the capping plane. • PRO_SKIRT_FILL_CAP_SURF—Specifies the closure from the capping surface. • PRO_SKIRT_FILL_NEAREST_PLANE—Specifies the closure from the nearest plane.
PRO_E_SKIRT_FILL_PLANE	PRO_VALUE_TYPE_SELECTION	Select a planar surface. This element is mandatory, if the element PRO_E_SKIRT_FILL_TYPE is set to the value PRO_SKIRT_FILL_MID_PLANE and PRO_SKIRT_FILL_CAP_PLANE.
PRO_E_SKIRT_FILL_SURF_COMPOUND	Compound	Select any surface. You can select all the surfaces except the reference model geometry. This element is mandatory, if the element PRO_E_SKIRT_FILL_TYPE is set to the value PRO_SKIRT_FILL_MID_SURF and PRO_SKIRT_FILL_CAP_SURF.
PRO_E_SKIRT_FILL_LOOP_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the loop offset. This element is used only when the element PRO_E_SKIRT_FILL_TYPE is set to the value PRO_SKIRT_FILL_MID_PLANE or PRO_SKIRT_FILL_CAP_PLANE.

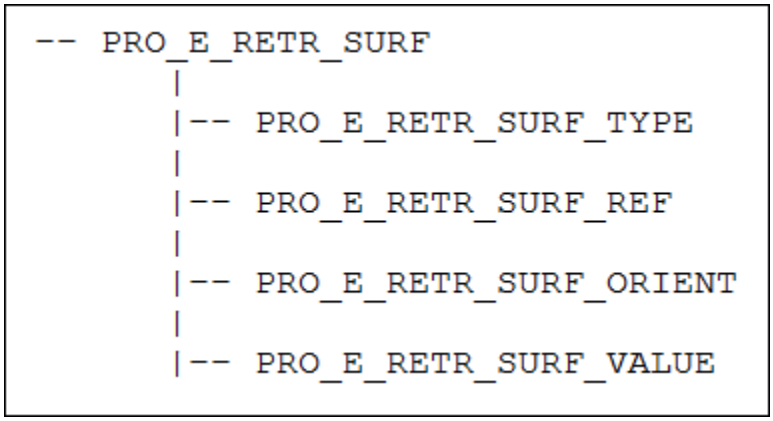
Sub-Element Trees: Creo NC Steps

This section describes how to construct and access the sub-element trees that are used in the creation of Creo NC features.

Retract Elements

The element PRO_E_RETR_SURF is documented in the header file ProMfgElemRetract.h, and is as shown in the following figure.

Element tree for PRO_E_RETR_SURF element



The following table lists the contents of PRO_E_RETR_SURF element.

Element ID	Data Type	Description
PRO_E_RETR_SURF_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of retract surface. The valid values for this element are: • PRO_RETR_SURF_UNDEFINED—Specifies that the retract surface type is not defined. This value is applicable only for manufacturing operations. • PRO_RETR_SURF_PLANE—Specifies that the retract surface defined is a planar surface. • PRO_RETR_SURF_CYLINDER—Specifies that the retract surface defined is a cylindrical surface. • PRO_RETR_SURF_SPHERE—Specifies that the retract surface defined is a spherical surface. • PRO_RETR_SURF_REVOLVED—Specifies that the retract surface defined is a revolved surface. **Note** The values PRO_RETR_SURF_CYLINDER, PRO_RETR_SURF_SPHERE, and PRO_RETR_SURF_REVOLVED are not applicable for 3-axis sequences and operations with 3-axis workcell.
PRO_E_RETR_SURF_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the selection of retract surface. The type of reference depends on values specified for the element PRO_E_RETR_SURF_TYPE. The valid values for this element are:

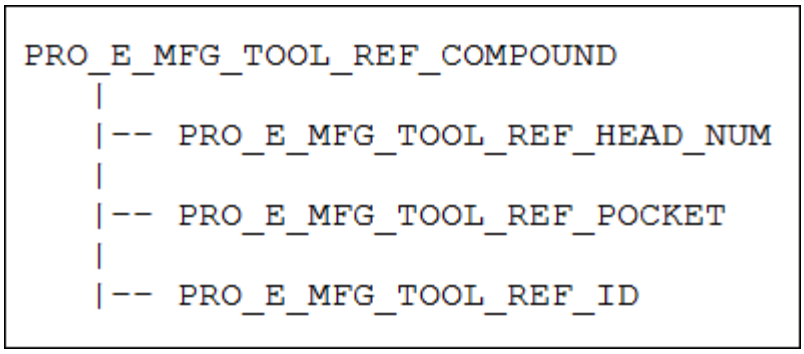
Element ID	Data Type	Description
		• Plane retract: Planar retract includes datum plane, planar surface, planar quilt and coordinate system. **Note** For 3-axis sequence (operation): ○ Planar surface must be normal to Z axis of the sequence (operation) coordinate system. ○ Z axis of selected datum coordinate system must be alligned with Z axis of sequence (operation) coordinate system. • Cylinder retract: Cylinder retract includes datum axis and coordinate system. • Sphere retract: It includes datum point and coordinate system. • Revolved surface retract: It includes revolved quilts.
PRO_E_RETR_SURF_ORIENT	PRO_VALUE_TYPE_INT	Specifies the orientation of the axis for the retract cylinder if coordinate system is selected as retract reference The valid values for this element are: • PRO_RETR_SURF_X_DIR • PRO_RETR_SURF_Y_DIR • PRO_RETR_SURF_Z_DIR **Note** This element is mandatory if coordinate system is selected for cylindrical retract reference and is ignored in all other cases.

Element ID	Data Type	Description
PRO_E_RETR_SURF_VALUE	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the offset value for the planar retract. For cylindrical and spherical retracts, this element specifies the value of the radius.

Tool Reference

The element PRO_E_MFG_TOOL_REF_COMPOUND is documented in the header file ProMfgElemToolRef.h, and is as shown in the following figure.

Element tree for PRO_E_MFG_TOOL_REF_COMPOUND element



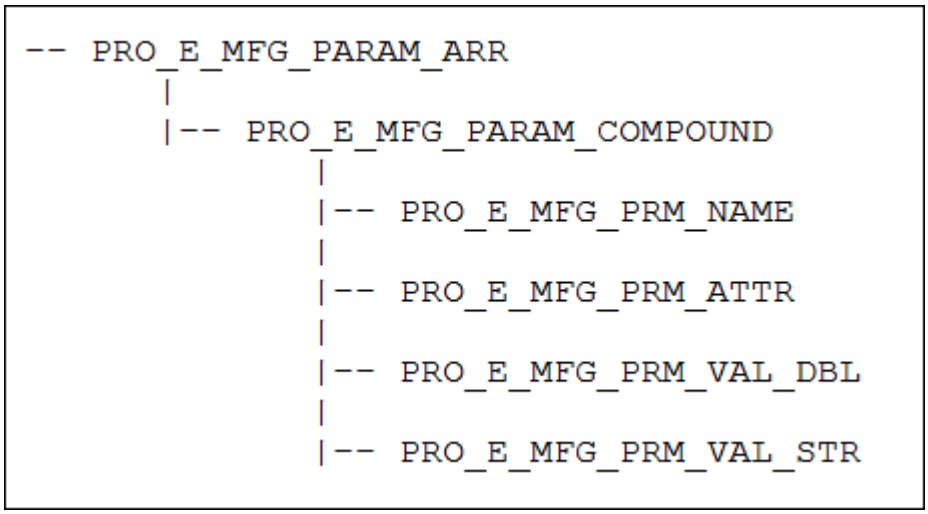
The following table lists the contents of PRO_E_MFG_TOOL_REF_COMPOUND element.

Element ID	Data Type	Description
PRO_E_MFG_TOOL_REF_HEAD_NUM	PRO_VALUE_TYPE_INT	Mandatory element. Head (turret) number. The valid values for this element are: • PRO_MFG_TOOL_HEAD_1—Specifies that the tool is placed in the head 1. • PRO_MFG_TOOL_HEAD_2—Specifies that the tool is placed in the head 2. • PRO_MFG_TOOL_HEAD_3—Specifies that the tool is placed in the head 3. • PRO_MFG_TOOL_HEAD_4—Specifies that the tool is placed in the head 4. **Note** You can specify the tools in head 1 for all types of workcells. Tools in head 2 can be specified for lathe and mill-turn workcells and tools in heads 3 and 4 can be specified for mill-turn workcells.
PRO_E_MFG_TOOL_REF_POCKET	PRO_VALUE_TYPE_INT	Mandatory element. This element defines the position of the tool in the turret head.
PRO_E_MFG_TOOL_REF_ID	PRO_VALUE_TYPE_WSTRING	Mandatory element. Specifies the tool Id.

Manufacturing Parameters

The element PRO_E_MFG_PARAM_ARR is documented in the header file ProMfgElemParam.h, and is as shown in the following figure.

Element tree for PRO_E_MFG_PARAM_ARR element



The element PRO_E_MFG_PARAM_ARR contains the compound element PRO_E_MFG_PARAM_COMPOUND that defines the name, attribute, and value of the parameter. You must define this element for parameters, such as, CUT_FEED and SPINDLE_SPEED, which do not have a default value. Refer to the Creo NC Manufacturing Help for more information on Creo NC Parameters.

The following table lists the contents of PRO_E_MFG_PARAM_ARR element.

Element ID	Data Type	Description
PRO_E_MFG_PRM_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the untranslated parameter name from the list predefined names. **Note** This element is a mandatory child of PRO_E_MFG_PARAM_COMPOUND element.
PRO_E_MFG_PRM_ATTR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the option to define logic for parameter value assignment. The values for this element are defined by ProMfgParamAttr. The valid values for this element are: • PRO_MFG_PRM_ATTR_DEFAULT—The value from PRO_E_MFG_PRM_VAL_DBL or PRO_E_MFG_PRM_VAL_STR is going to be assigned to the corresponding manufacturing parameter. • PRO_MFG_PRM_ATTR_AUTOMATIC—Default value

Element ID	Data Type	Description
		(supplied by Creo NC) is going to be assigned to the corresponding manufacturing parameter. Use PRO_MFG_PRM_ATTR_DEFAULT to set param value to dash (if applicable). Value from PRO_E_MFG_PRM_VAL_DBL or PRO_E_MFG_PRM_VAL_STR is ignored. • PRO_MFG_PRM_ATTR_MODIFIED—Value from site or parent sequence (for tool motion parameters) is going to be assigned to the corresponding manufacturing parameter. Value from PRO_E_MFG_PRM_VAL_DBL or PRO_E_MFG_PRM_VAL_STR is going to be used only if site with corresponding name does not exist in manufacturing model (The site name is specified by PRO_E_MFG_PARAM_SITE_NAME element in sequence elem tree). • PRO_MFG_PRM_ATTR_INHERITED—Value will be assigned automatically: either from site (if exists), or from system defaults. The Value from PRO_E_MFG_PRM_VAL_DBL or PRO_E_MFG_PRM_VAL_STR is ignored. **Note** This element is a mandatory child of PRO_E_MFG_PARAM_COMPOUND element.
PRO_E_MFG_PRM_VAL_DBL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of type double. **Note** This element is Mandatory for Double data type parameters with attribute set to PRO_MFG_PRM_ATTR_MODIFIED. Ignored for other data types.

Element ID	Data Type	Description
PRO_E_MFG_PRM_VAL_STR	PRO_VALUE_TYPE_WSTRING	Specifies the value of type string. **Note** Mandatory for String data type parameters with attribute set to PRO_MFG_PRM_ATTR_MODIFIED. Ignored for other data types.

Surface Collection with Mill Window

The element PRO_E_MFG_CMP_MILL_WIND is documented in the header file ProMfgElemMachWindow.h, and is as shown in the following figure.

Element tree for PRO_E_MFG_CMP_MILL_WIND element

```
-- PRO_E_MFG_CMP_MILL_WIND
       |
       |-- PRO_E_MFG_MILL_WIND
       |
       |-- PRO_E_MFG_CMP_CLOSED_LOOPS
       |
       |-- PRO_E_MFG_EXCL_SRF_COLL
       |         |
       |         |-- PRO_E_STD_SURF_COLLECTION_APPL
       |
       |-- PRO_E_MFG_SURF_SIDE_COMPOUND
       |         |
       |         |-- PRO_E_MFG_SURF_SIDE_TOLERANCE
       |         |
       |         |-- PRO_E_MFG_SURF_SIDE_FLIP_QUILTS
       |
```

The following table lists the contents of PRO_E_MFG_CMP_MILL_WIND element

Element ID	Data Type	Description
PRO_E_MFG_MILL_WIND	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the selection of mill window feature.
PRO_E_MFG_CMP_CLOSED_LOOPS	Compound	Optional element. Specifies closed loop compound specification.
PRO_E_MFG_EXCL_SRF_COLL	Compound :D	Optional element. Defines the

Element ID	Data Type	Description
		excluded surfaces. **Note** Specify a value for this element only for finishing and corner finishing sequences.
PRO_E_STD_SURF_COLLECTION_APPL	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the surface collection.
PRO_E_MFG_SURF_SIDE_COMPOUND	Compound	Optional element. Specifies the surface side compound specification. This element is used as a sub element of PRO_E_MFG_CMP_MILL_WIND only when you click **Adjust geometry collection within window** and select **Quilt** during the Mill Window feature definition in the Creo Parametric user interface.
PRO_E_MFG_SURF_SIDE_TOLERANCE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the tolerance used for the grouping of quilts.
PRO_E_MFG_SURF_SIDE_FLIP_QUILTS	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of quilts with changed normal. This element supports multiple selections.

PRO_E_MFG_CMP_CLOSED_LOOPS Element

The elements under the compound element PRO_E_MFG_CMP_CLOSED_LOOPS are described in this section .

Element tree for PRO_E_MFG_CMP_CLOSED_LOOPS element

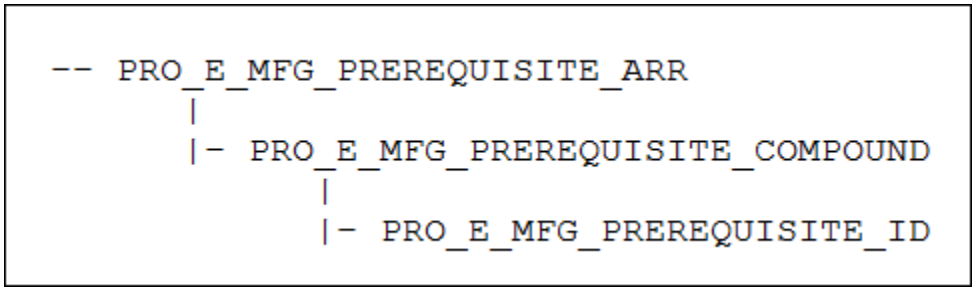
```
|--PRO_E_MFG_CMP_CLOSED_LOOPS
|    |
|    |-- PRO_E_MFG_CLOSED_LOOP_ARR
|          |
|          |-- PRO_E_MFG_CLOSED_LOOP_REF_ITEM
|                |
|                |-- PRO_E_STD_CURVE_COLLECTION_APPL
|                |
|                |-- PRO_E_MFG_CLOSED_LOOP_ID
```

Element ID	Data Type	Description
PRO_E_MFG_CLOSED_LOOP_ARR	Array	Optional element. Specifies an array of closed loop specifications.
PRO_E_MFG_CLOSED_LOOP_REF_ITEM	Compound	Optional element. Specifies the closed loop specification.
PRO_E_STD_CURVE_COLLECTION_APPL	PRO_VALUE_TYPE_POINTER	Specifies the chain collection. **Note** This element is a mandatory child of PRO_E_MFG_CLOSED_LOOP_REF_ITEM compound element.

Sequence Prerequisites

The element PRO_E_MFG_PREREQUISITE_ARR is documented in the header file ProMfgElemPrerequisite.h, and is shown in the following figure.

Element tree for PRO_E_MFG_PREREQUISITE_ARR element



The following table lists the contents of PRO_E_MFG_PREREQUISITE_ARR element.

Element ID	Data Type	Description
PRO_E_MFG_PREREQUISITE_COMPOUND	Compound	Compound element. A compound element which defines the prerequisites for the step.
PRO_E_MFG_PREREQUISITE_ID	PRO_VALUE_TYPE_INT	Mandatory child of PRO_E_MFG_PREREQUISITE_COMPOUND element. Specifies the Id of the prerequisite sequence.

Element Trees: Tool Setup

The Tool Setup Feature Element Tree:

The element tree for the milling roughing sequence is documented in the header file `ProMfgElemToolSetup.h`, and is as shown in the following figure:

Element Tree for Tool Setup feature

```
-- PRO_E_MFG_WCELL_TOOL_SETUP_ARR
    |
    |- PRO_E_MFG_WCELL_TOOL_SETUP
        |
        |- PRO_E_MFG_WCELL_TOOL_POCKET_NUM
        |
        |- PRO_E_MFG_WCELL_TOOL_ID
        |
        |- PRO_E_MFG_WCELL_TOOL_OUTPUT_TIP
        |
        |- PRO_E_MFG_TOOL_TIP_ARR
            |
            |- PRO_E_MFG_TOOL_TIP_COMPOUND
                |
                |- PRO_E_MFG_TOOL_TIP_REGISTER
                |
                |- PRO_E_MFG_TOOL_TIP_COMMENT
                |
                |- PRO_E_MFG_TOOL_TIP_OFFSET_Z
                |
                |- PRO_E_MFG_TOOL_TIP_OFFSET_X
                |
                |- PRO_E_MFG_TOOL_TIP_OFFSET_ANGLE
                |
                |- PRO_E_MFG_TOOL_TIP_FLASH_OPT
                |
                |- PRO_E_MFG_TOOL_TIP_FLSH_REGISTER
                |
                |- PRO_E_MFG_TOOL_TIP_FLSH_COMMENT
                |
                |- PRO_E_MFG_TOOL_TIP_FLSH_OFFSET_Z
                |
                |- PRO_E_MFG_TOOL_TIP_FLSH_OFFSET_X
```

Element ID	Data Type	Description
PRO_E_MFG_WCELL_TOOL_SETUP_ARR	Array	Optional element. Specifies the tool setup array.
PRO_E_MFG_WCELL_TOOL_SETUP	Compound	Optional element. Specifies the tool setup compound specification.
PRO_E_MFG_WCELL_TOOL_POCKET_NUM	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool position in head (turret).
PRO_E_MFG_WCELL_TOOL_ID	PRO_VALUE_TYPE_WSTRING	Mandatory feature. Specifies the tool ID. **Note** Tool with such ID should exist in manufacturing model.
PRO_E_MFG_WCELL_TOOL_OUTPUT_TIP	PRO_VALUE_TYPE_INT	Optional element, if not defined or if the value is set to 1. Specifies the tip number. **Note** The tool tip number should not be greater than the number of children in PRO_E_MFG_TOOL_TIP_ARR.
PRO_E_MFG_TOOL_TIP_ARR	Array	Optional element. Specifies an array of tips.
PRO_E_MFG_TOOL_TIP_COMPOUND	Compound	Optional element. This compound element defines the tip specification.
PRO_E_MFG_TOOL_TIP_REGISTER	PRO_VALUE_TYPE_INT	Optional element. Specifies the tip register number.
PRO_E_MFG_TOOL_TIP_COMMENT	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the tip comment.
PRO_E_MFG_TOOL_TIP_OFFSET_Z	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset value for the tool tip in the Z-direction.
PRO_E_MFG_TOOL_TIP_OFFSET_X	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset value for the tool tip in the X-direction.
PRO_E_MFG_TOOL_TIP_OFFSET_ANGLE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the tip angle offset value.
PRO_E_MFG_TOOL_TIP_FLASH_OPT	PRO_VALUE_TYPE_INT	Optional element, if the element is not defined or the value of the element is set to PRO_B_FALSE. Specifies the enabling/disabling of the flash option of the tool tip. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_B_TRUE—Enables flashing capability. • PRO_B_FALSE—Disables flashing capability.
PRO_E_MFG_TOOL_TIP_FLSH_REGISTER	PRO_VALUE_TYPE_INT	Optional element. Specifies the register number for alternate tip for the flash tool. **Note** This element is ignored if the element PRO_E_MFG_TOOL_TIP_FLASH_OPT is set to PRO_B_FALSE.
PRO_E_MFG_TOOL_TIP_FLSH_COMMENT	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the flash tool alternate tip comments. **Note** This element is ignored if the element PRO_E_MFG_TOOL_TIP_FLASH_OPT is set to PRO_B_FALSE
PRO_E_MFG_TOOL_TIP_FLSH_OFFSET_Z	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the flash tool alternate tip Z offset. **Note** This element is ignored if the element PRO_E_MFG_TOOL_TIP_FLASH_OPT is set to PRO_B_FALSE
PRO_E_MFG_TOOL_TIP_FLSH_OFFSET_X	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the flash tool alternate tip X offset. **Note** This element is ignored if the element PRO_E_MFG_TOOL_TIP_FLASH_OPT is set to PRO_B_FALSE

Element Trees: CMM Probe Setup

The CMM probe Setup Element Tree:

The element tree for the CMM probe setup is documented in the header file `ProMfgElemToolSetupCmm.h`, and is as shown in the following figure:

Element Tree for CMM Probe Setup

```
-- PRO_E_MFG_CMM_TOOL_SETUP_ARR
       |
       |- PRO_E_MFG_CMM_TOOL_SETUP
            |
            |- PRO_E_MFG_CMM_TOOL_POCKET_NUM
            |
            |- PRO_E_MFG_CMM_TOOL_TOOL_ID
            |
            |- PRO_E_MFG_CMM_TOOL_TIP_NUM
            |
            |- PRO_E_MFG_CMM_TOOL_REGISTER
            |
            |- PRO_E_MFG_CMM_TOOL_COMMENT
            |
            |- PRO_E_MFG_CMM_TOOL_PITCH_ANGLE
            |
            |- PRO_E_MFG_CMM_TOOL_ROLL_ANGLE
```

The following table describes the elements in the element tree for the CMM probe setup feature.

Element ID	Data Type	Description
`PRO_E_MFG_CMM_TOOL_SETUP_ARR`	`Array`	Optional element. Specifies the CMM probes setup array.
`PRO_E_MFG_CMM_TOOL_SETUP`	`Compound`	Optional element. This compound element defines the probe setup compound specification.
`PRO_E_MFG_CMM_TOOL_POCKET_NUM`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies the probe position in tool magazine.
`PRO_E_MFG_CMM_TOOL_TOOL_ID`	`PRO_VALUE_TYPE_WSTRING`	Mandatory element. Specifies the Probe ID. **Note** Tool with such ID should exist in manufacturing model.
`PRO_E_MFG_CMM_TOOL_TIP_NUM`	`PRO_VALUE_TYPE_INT`	Optional element, if not defined or if the value is set to 1. Specifies the tip number by identifying it from the array.

Element ID	Data Type	Description
PRO_E_MFG_CMM_TOOL_REGISTER	PRO_VALUE_TYPE_INT	Optional element. Specifies the register number.
PRO_E_MFG_CMM_TOOL_COMMENT	PRO_VALUE_TYPE_WSTRING	Optional element. Specifies the probe comments.
PRO_E_MFG_CMM_TOOL_PITCH_ANGLE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the pitch angle value.
PRO_E_MFG_CMM_TOOL_ROLL_ANGLE	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the roll angle value.

Checking Surfaces

The element PRO_E_CHECK_SURF_COLL is documented in the header file ProMfgElemCheckSurf.h, and is shown in the following figure.

Element tree for PRO_E_CHECK_SURF_COLL element

```
-- PRO_E_CHECK_SURF_COLL
       |
       |-- PRO_E_ADD_REF_PARTS
       |
       |-- PRO_E_USE_MILL_STK
       |
       |-- PRO_E_STD_SURF_COLLECTION_APPL
```

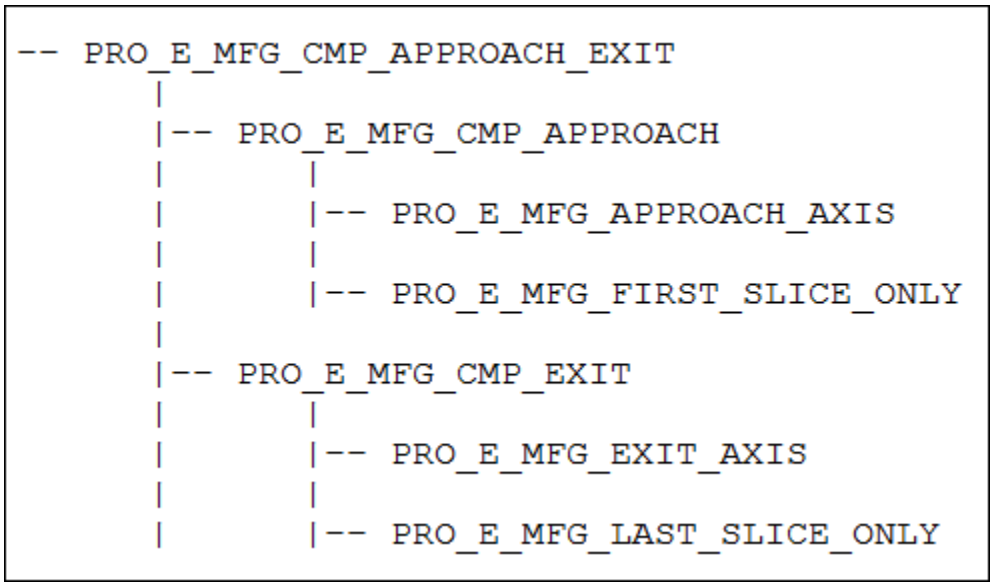
The following table lists the contents of PRO_E_CHECK_SURF_COLL element.

Element ID	Data Type	Description
PRO_E_ADD_REF_PARTS	PRO_VALUE_TYPE_INTEGER	Optional element. This element is used to check the reference parts for collisions. The valid values for this element are: • TRUE—All reference parts surfaces will be checked for collisions. • FALSE—Reference parts surfaces will not be added for collision checking. **Note** The FALSE value will be used if element does not exist.
PRO_E_USE_MILL_STK	PRO_VALUE_TYPE_INTEGER	Optional element. This element is used to apply the stock allowance parameters of mill surface to the surfaces being checked. The valid values for this element are: • TRUE— Mill surface stock allowance parameter will be applied to check surfaces. Value of stock allowance is defined by sequence manufacturing parameter. • FALSE—Mill surface stock allowance parameter will not be applied. **Note** The FALSE value will be used if element does not exist.
PRO_E_STD_SURF_ COLLECTION_APPL	Surface Collection	Optional element. Specifies the collection of selected surfaces to be checked for collisions.

Approach and Exit

The element tree for the approach and exit parameters is defined in the header file `ProMfgElemApproachExit.h`, and is as shown in the following figure:

Element tree for PRO_E_MFG_CMP_APPROACH_EXIT element



The following table lists the contents of `PRO_E_MFG_CMP_APPROACH_EXIT` element.

Element ID	Data Type	Description
`PRO_E_MFG_CMP_APPROACH`	`Compound`	Optional element. This compound element specifies approach compound. It combines approach axis and first slice only.
`PRO_E_MFG_APPROACH_AXIS`	`PRO_VALUE_TYPE_SELECTION`	Optional element. Specifies the selection of a vertical datum axis.
`PRO_E_MFG_FIRST_SLICE_ONLY`	`PRO_VALUE_TYPE_INT`	Specifies the flag value for the approach motion. The valid values for this element are: • True • False **Note** This element is mandatory if the element `PRO_E_MFG_APPROACH_AXIS` is set. It is not used otherwise.
`PRO_E_MFG_CMP_EXIT`	`Compound`	This compound element specifies an exit compound. It combines exit axis and last slice only.

Element ID	Data Type	Description
PRO_E_MFG_EXIT_AXIS	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of a vertical datum axis.
PRO_E_MFG_LAST_SLICE_ONLY	PRO_VALUE_TYPE_INT	Specifies the flag value for the exit motion. The valid values for this element are: • True • False **Note** This element is mandatory if the element PRO_E_MFG_EXIT_AXIS is set. It is not used otherwise.

Spindle Types

This section describes the types of spindles, which can be used while creating a sequence feature. The types of spindles are defined by the enumerated type ProSubSpindleOpt and are as follows:

- PRO_MFG_MAIN_SPINDLE—Specifies that the sequence feature is created for the part in the main spindle.
- PRO_MFG_SUB_SPINDLE—Specifies that the sequence feature is created for the part in the sub–spindle.

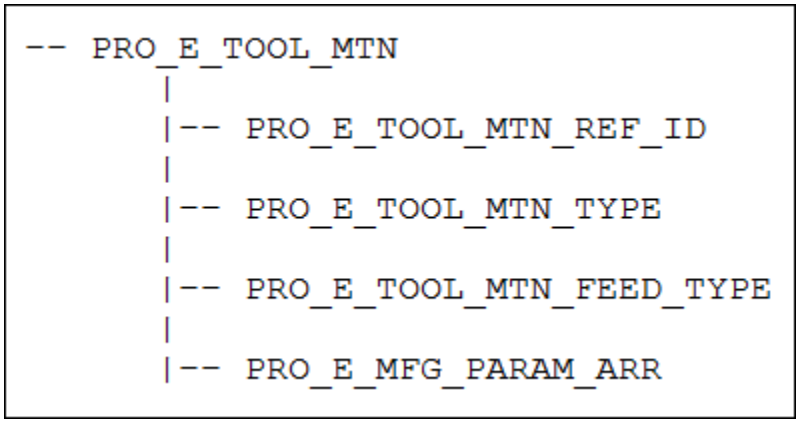
Two parts can be machined during the same operation using the main spindle and the sub spindle.

Tool Motions

This section describes how to construct and access the sub-element trees for various Tool Motion types that are used in the creation of Creo NC features.

Approach Along Tool Axis

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnAlongAxisAppr.h, and is as shown in the following figure:

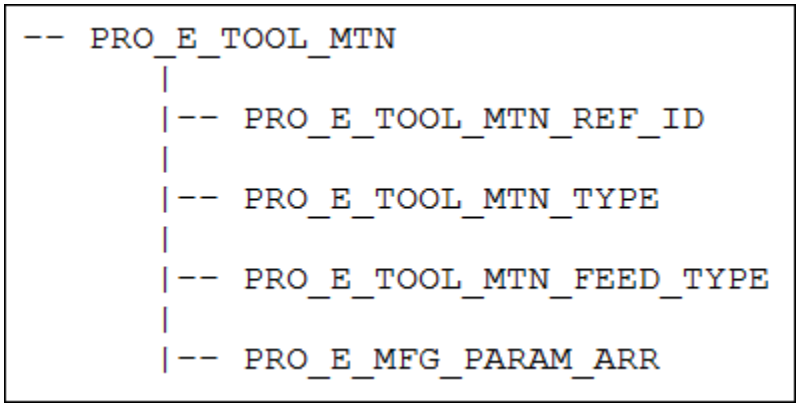


The following table describes the elements in the element tree for the approach along tool axis feature.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_TYPE_ALONG_AXIS_APPROACH.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE—Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Exit Along Tool Axis

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnAlongAxisExit.h, and is as shown in the following figure:



The following table describes the elements in the element tree for the approach along tool axis feature.

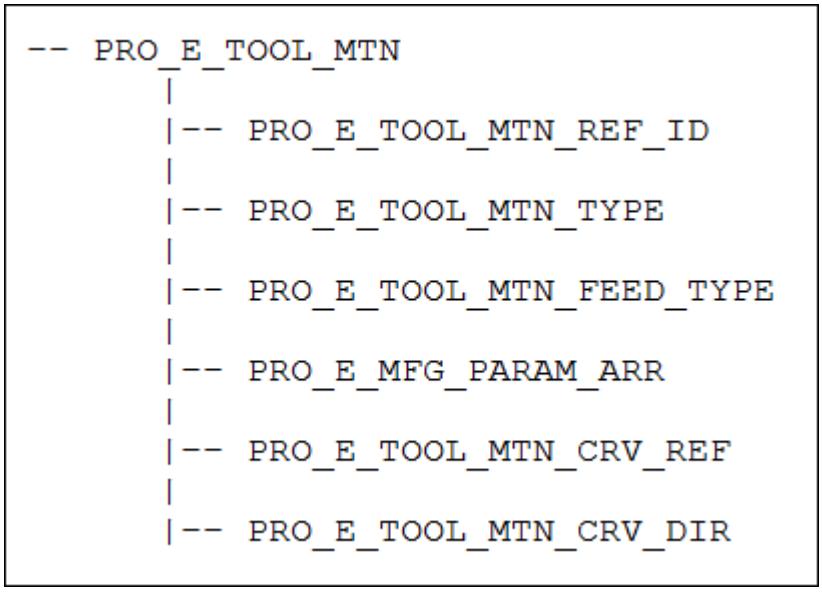
Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_TYPE_ALONG_AXIS_EXIT.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_RETRACT— Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_EXIT— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Follow Curve

The element PRO_E_TOOL_MTN is a compound element that allows you to specify the tool motion parameters.

This element is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnFollowCrv.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

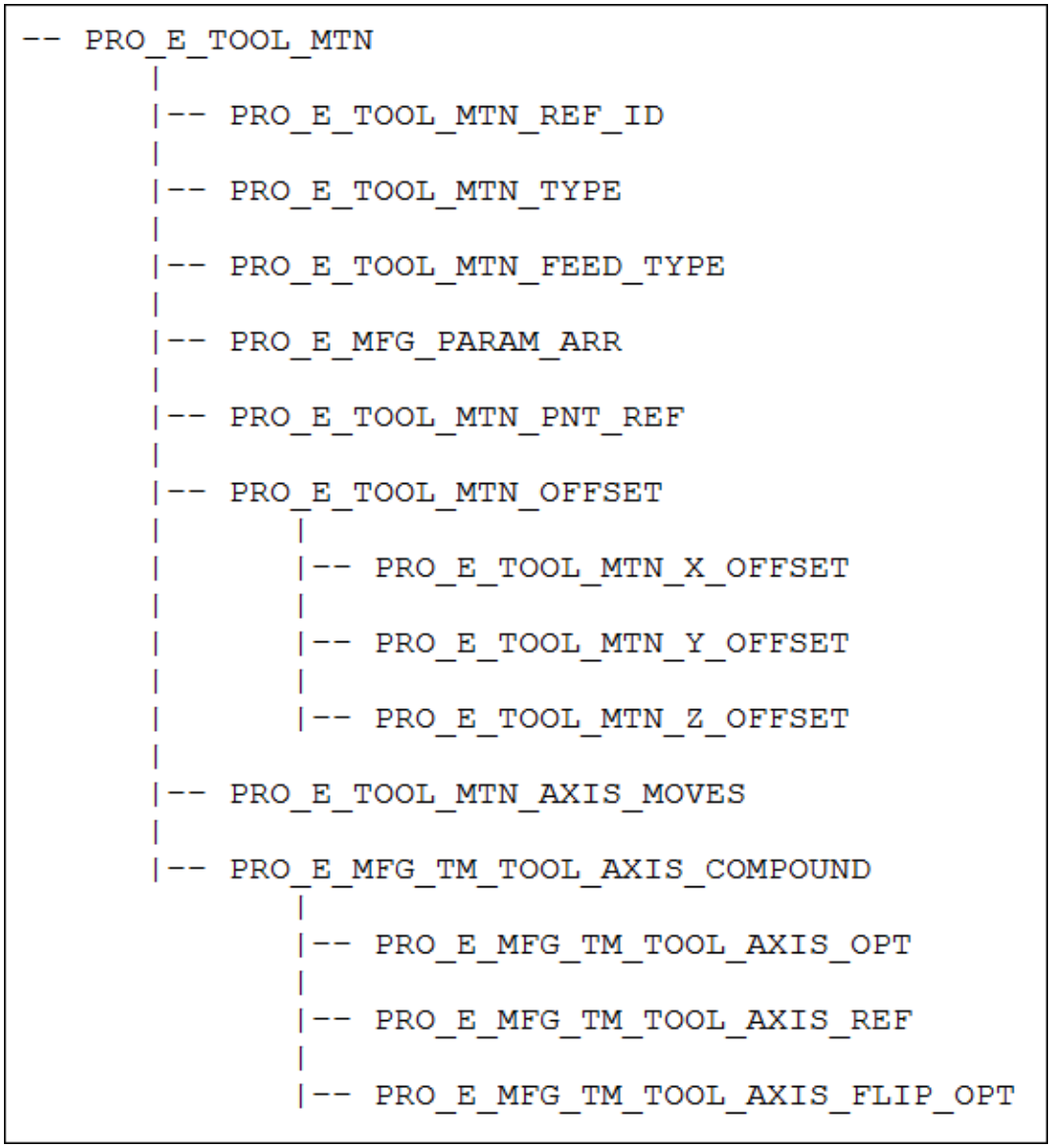
Element ID	Data Type	Description
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the tool motion type . The valid value for this element is `PRO_TM_TYPE_FOLLOW_CURVE`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are: • `PRO_TM_FEED_FREE`— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • `PRO_TM_FEED_CUT`— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • `PRO_TM_FEED_APPROACH`—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece. • `PRO_TM_FEED_EXIT`— Specifies an exit feed type. Exit feed specifies the rate at

Element ID	Data Type	Description
		which the tool leaves the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_CRV_REF	PRO_VALUE_TYPE_ SELECTION	Mandatory element. Specifies the selection of the curve feature.
PRO_E_TOOL_MTN_CRV_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of the working for the curve feature. The direction is defined by the enumerated data type ProMfgCrvDir in ProMfgOptions.h. The valid values for this element are: • PRO_MFG_DIR_OPPOSITE • PRO_MFG_DIR_SAME

Tool Motion — Go To Point

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGotoPnt.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

Element ID	Data Type	Description
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies the tool motion type . The valid value for this element is `PRO_TM_TYPE_GOTO_POINT`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The

Element ID	Data Type	Description
		valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE—Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the workpiece. • PRO_TM_FEED_RETRACT—Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece. • PRO_TM_FEED_EXIT—Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece. • PRO_TM_FEED_THREAD—Specifies a thread feed type. Thread feed specifies the feed rate for the tapping step.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_PNT_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the datum point reference.

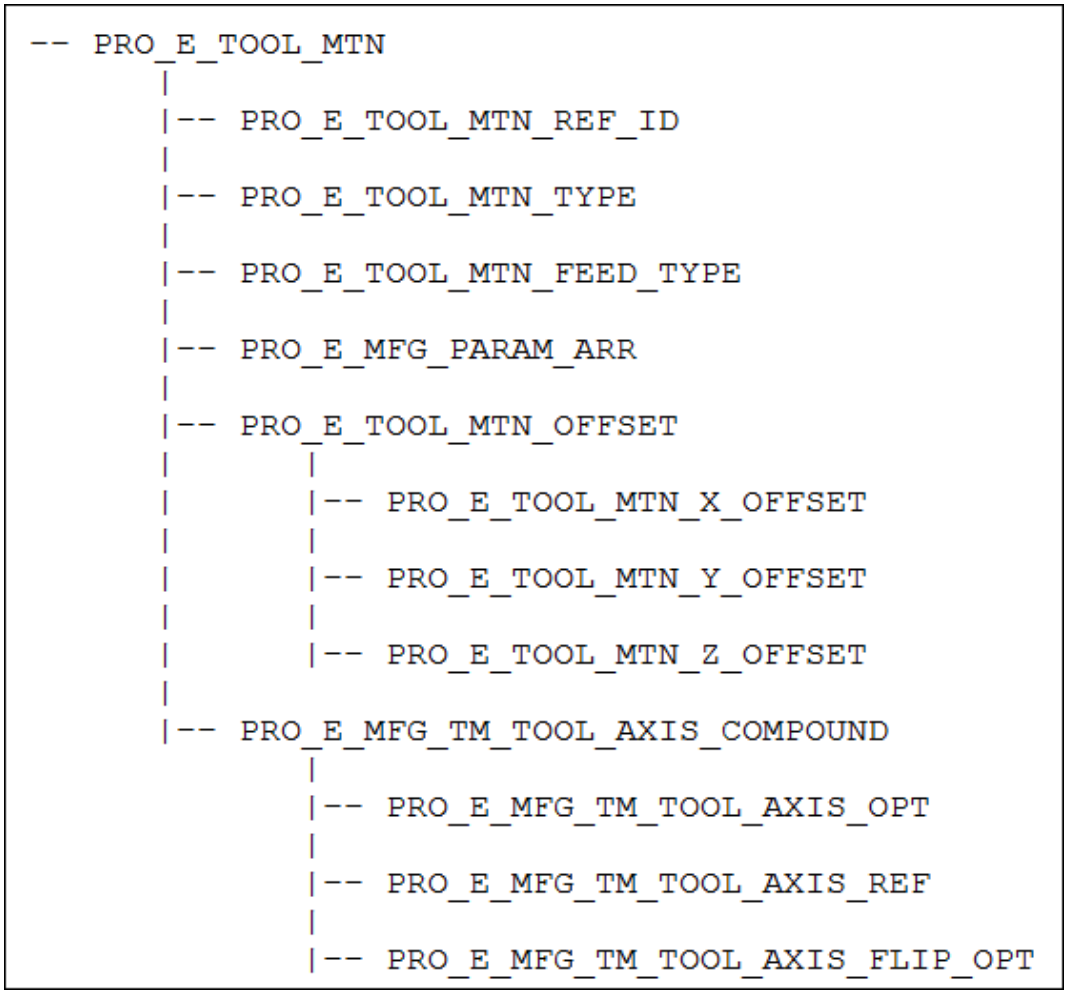
Element ID	Data Type	Description
PRO_E_TOOL_MTN_OFFSET	Compound	Optional element. This compound element specifies the definition for the various tool motion offset parameters.
PRO_E_TOOL_MTN_X_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along X-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Y_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Y-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Z-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE
PRO_E_TOOL_MTN_AXIS_MOVES	PRO_VALUE_TYPE_INT	Mandatory if the element PRO_E_TOOL_MTN_OFFSET is defined. Specifies the attributes for the axis moves.
PRO_E_MFG_TM_TOOL_AXIS_COMPOUND	Compound	Optional element. This compound element specifies the tool axis.
PRO_E_MFG_TM_TOOL_AXIS_OPT	PRO_VALUE_TYPE_INT	Optional element. This element specifies the tool axis options using the enumerated type ProTmToolAxisOpt. The valid values for this element are: • PRO_TM_ALONG_Z • PRO_TM_USE_PREV • PRO_TM_AXIS_SEL

Element ID	Data Type	Description
PRO_E_MFG_TM_TOOL_AXIS_REF	PRO_VALUE_TYPE_SELECTION	This element specifies the axis selection. **Note** This element is mandatory if the element PRO_E_MFG_TM_TOOL_AXIS_OPT is set to PRO_TM_AXIS_SEL.
PRO_E_MFG_TM_TOOL_AXIS_FLIP_OPT	PRO_VALUE_TYPE_INT	Specifies the flip options. The valid values for this element are: • TRUE • FALSE **Note** This element is mandatory if the element PRO_E_MFG_TM_TOOL_AXIS_OPT is set to PRO_TM_AXIS_SEL.

Tool Motion — Go Delta

The PRO_E_TOOL_MTN element is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGoDelta.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

Element ID	Data Type	Description
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the Tool motion type . The valid value for this element is `TPRO_TM_TYPE_GO_DELTA`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are:

Element ID	Data Type	Description
		• PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE—Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the workpiece. • PRO_TM_FEED_RETRACT—Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece. • PRO_TM_FEED_EXIT—Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece. • PRO_TM_FEED_THREAD—Specifies a thread feed type. Thread feed specifies the feed rate for the tapping step.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_OFFSET	Compound	Optional element. This compound

Element ID	Data Type	Description
		element specifies the offset value. **Note** At least one of elements below must have a non zero value. • PRO_E_TOOL_MTN_X_OFFSET • PRO_E_TOOL_MTN_Y_OFFSET • PRO_E_TOOL_MTN_Z_OFFSET
PRO_E_TOOL_MTN_X_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along X-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Y_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Y-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Z-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE
PRO_E_MFG_TM_TOOL_AXIS_COMPOUND	Compound	Optional element. This compound element specifies the tool axis.
PRO_E_MFG_TM_TOOL_AXIS_OPT	PRO_VALUE_TYPE_INT	Optional element. This element specifies the tool axis options. The valid values for this element are: • PRO_TM_ALONG_Z • PRO_TM_USE_PREV • PRO_TM_AXIS_SEL

Element ID	Data Type	Description
PRO_E_MFG_TM_TOOL_AXIS_REF	PRO_VALUE_TYPE_SELECTION	This element specifies the axis selection. **Note** This element is mandatory if the element PRO_E_MFG_TM_TOOL_AXIS_OPT is set to PRO_TM_AXIS_SEL.
PRO_E_MFG_TM_TOOL_AXIS_FLIP_OPT	PRO_VALUE_TYPE_INT	Specifies the flip options. The valid values for this element are: • TRUE • FALSE **Note** This element is mandatory if the element PRO_E_MFG_TM_TOOL_AXIS_OPT is set to PRO_TM_AXIS_SEL.

Tool Motion — Go Home

The PRO_E_TOOL_MTN element is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGoHome.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_TOOL_MTN_FEED_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
       |
       |-- PRO_E_TOOL_MTN_AXIS_REF
```

The following table lists the contents of PRO_E_TOOL_MTN element.

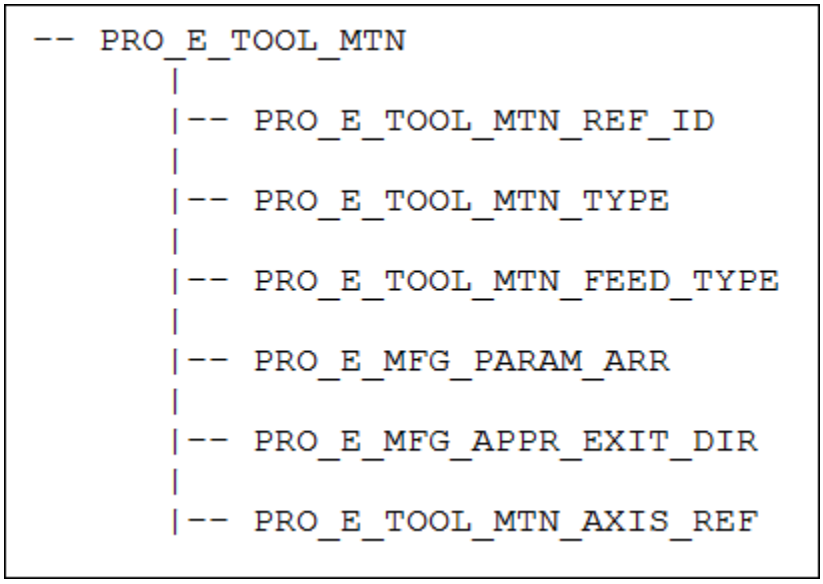
Element ID	Data Type	Description
PRO_E_TOOL_MTN	Compound	Compound element. This element specifies the tool motion definition.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_RETRACT— Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_EXIT— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_GOHOME. The value for this element is defined by ProTmType.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_ SELECTION	Specifies the axis selection. This element is optional for 5-axis Creo NC sequences. **Note** By default, Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences, you can select an alternative axis.

Tool Motion — Lead In

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnLeadIn.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

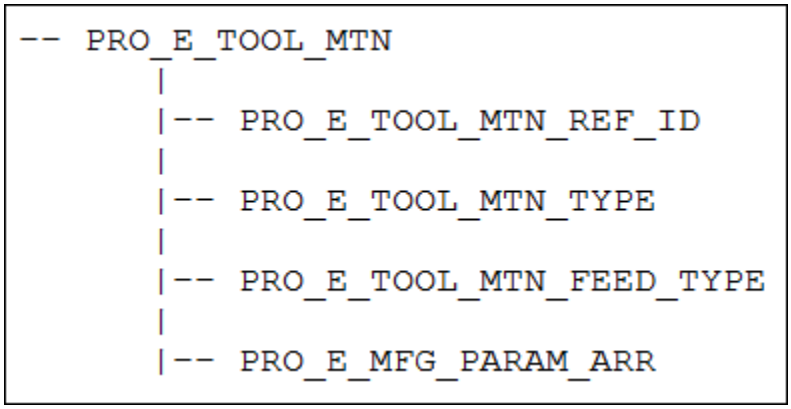
Element ID	Data Type	Description
`PRO_E_TOOL_MTN`	`Compound`	Compound element. This element specifies the tool motion definition.
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the tool motion type. The valid value for this element is `PRO_TM_TYPE_LEAD_IN`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are: • `PRO_TM_FEED_FREE`—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • `PRO_TM_FEED_CUT`—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • `PRO_TM_FEED_APPROACH`—Specifies an approach feed type. Approach feed specifies the rate at which

Element ID	Data Type	Description
		the tool approaches the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The values which are used to define this element are: • ENTRY_ANGLE • TANGENT_LEAD_STEP • NORMAL_LEAD_STEP • LEAD_RADIUS The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction for the normal approach type of tool exit. The direction is defined by the enumerated data type ProTmSideDirin ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection. **Note** This element is optional for 5-axis Creo NC sequences. By default Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences a user can select an alternative axis.

Tool Motion — Go Retract

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGoRetr.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

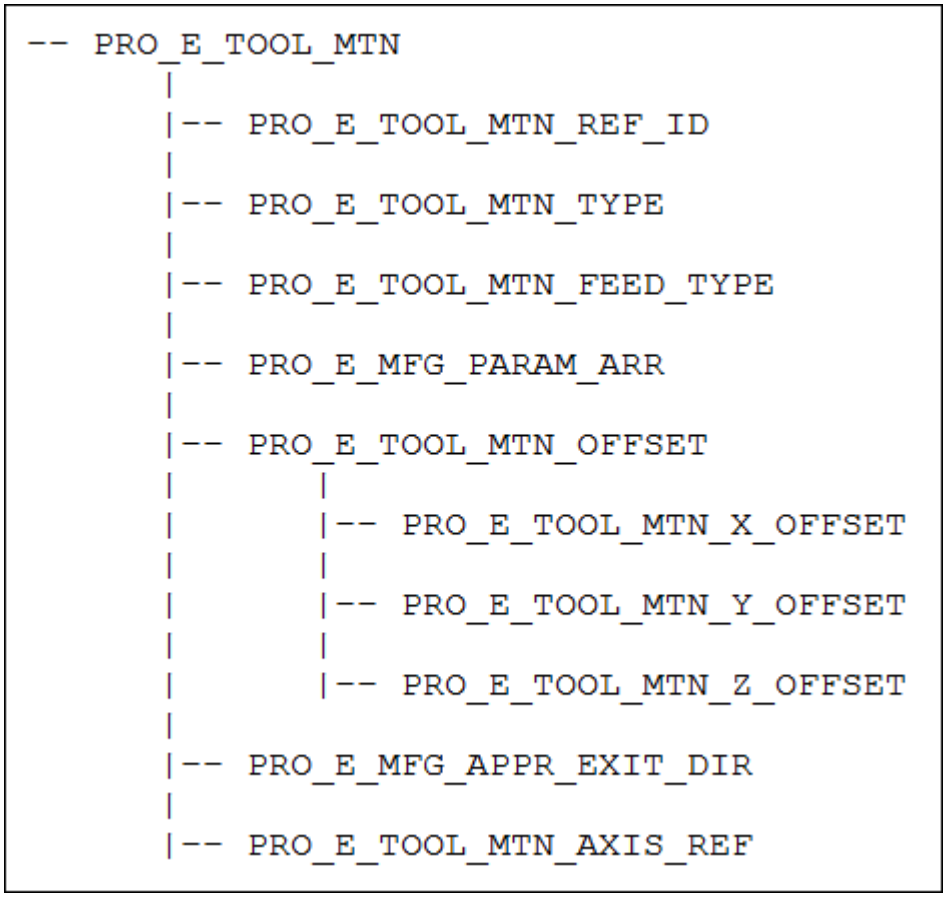
Element ID	Data Type	Description
`PRO_E_TOOL_MTN`	`Compound`	Compound element. This element specifies Tool motion definition.
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the Tool motion type . The valid value for this element is `PRO_TM_TYPE_GO_RETRACT`. The value for this element is defined by `ProTmType`.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_RETRACT—Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_EXIT—Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Normal Approach

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnNormAppr.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

Element ID	Data Type	Description
`PRO_E_TOOL_MTN`	`Compound`	Compound element. This element specifies the tool motion definition.
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the tool motion type. The valid value for this element is `PRO_TM_TYPE_NORMAL_APPROACH`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are:

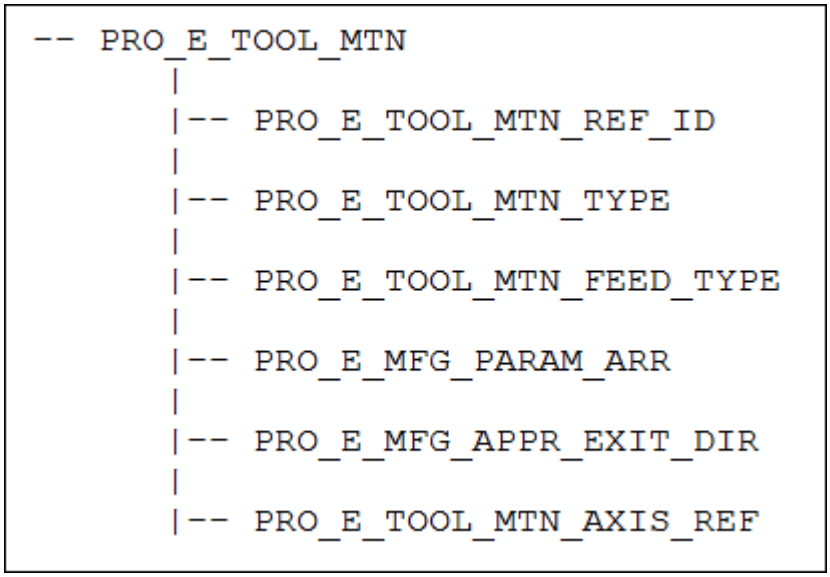
Element ID	Data Type	Description
		• PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element APPROACH_DIST is used to specify the approach distance. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677for more information on the element tree.
PRO_E_TOOL_MTN_OFFSET	Compound	Optional element. This compound element specifies the offset value.
PRO_E_TOOL_MTN_X_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along X-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Y_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Y-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Z-axis. This element can range from negative to positive values. The valid range values for this element are from

Element ID	Data Type	Description
		-MAX_DIM_VALUE to MAX_DIM_VALUE
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction for the normal approach type of tool exit. The direction is defined by the enumerated data type ProTmSideDir in ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection. **Note** This element is optional for 5-axis Creo NC sequences. By default Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences a user can select an alternative axis.

Tool Motion — Normal Exit

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnNormExit.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

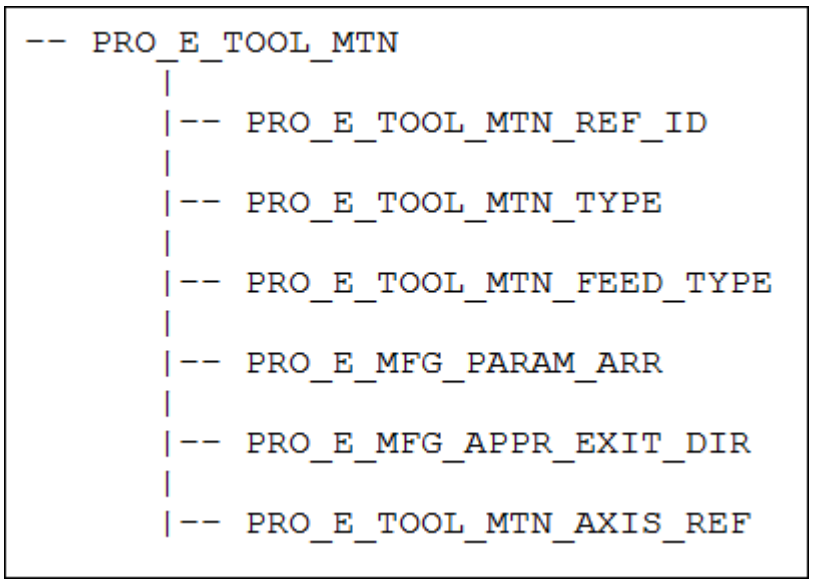
Element ID	Data Type	Description
`PRO_E_TOOL_MTN`	`Compound`	Compound element. This element specifies the Tool motion definition.
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Tool motion type . The valid value for this element is `PRO_TM_TYPE_NORMAL_EXIT`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are: • `PRO_TM_FEED_FREE`— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • `PRO_TM_FEED_CUT`— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • `PRO_TM_FEED_EXIT`— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element EXIT_DISTANCE is used to specify the exit distance. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of orientation for the approach or exit tool motion. The direction of the orientation is defined by the enumerated data type ProTmSideDir in ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection. **Note** This element is optional for 5-axis Creo NC sequences. By default Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences a user can select an alternative axis.

Tool Motion — Lead Out

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnLeadOut.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

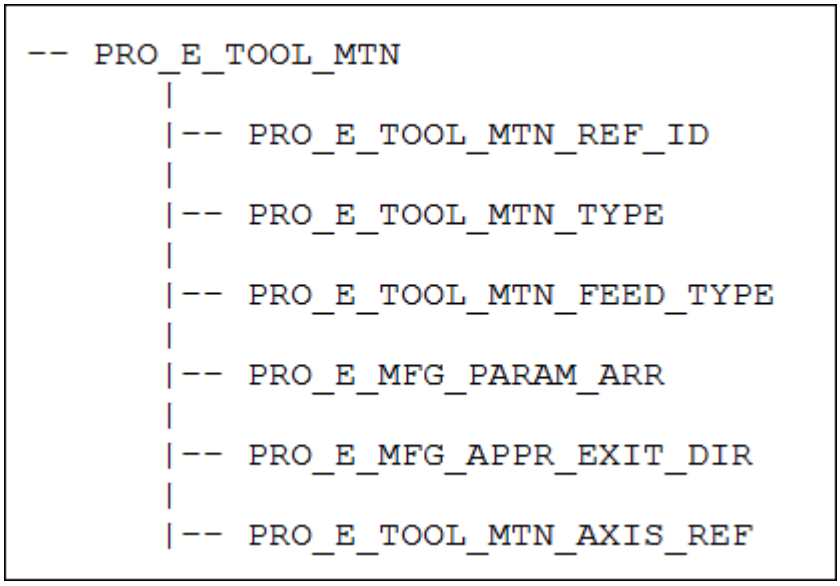
Element ID	Data Type	Description
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the tool motion type . The valid value for this element is `PRO_TM_TYPE_LEAD_OUT`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are: • `PRO_TM_FEED_FREE`— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • `PRO_TM_FEED_CUT`— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • `PRO_TM_FEED_EXIT`— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
`PRO_E_MFG_PARAM_ARR`	`Array`	Mandatory element. Specifies an array of manufacturing

Element ID	Data Type	Description
		parameters. The following parameters are used to define the Lead Out Motion: • EXIT_ANGLE • TANGENT_LEAD_STEP • NORMAL_LEAD_STEP • LEAD_RADIUS The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of orientation for approach/exit tool motion. The direction of orientation is defined by the enumerated data type ProTmSideDir in ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection. **Note** This element is optional for 5-axis Creo NC sequences. By default Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences a user can select an alternative axis.

Tool Motion — Helical Approach

The element PRO_E_TOOL_MTN is documented in the header file ProMfgElemToolMtnHelAppr.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

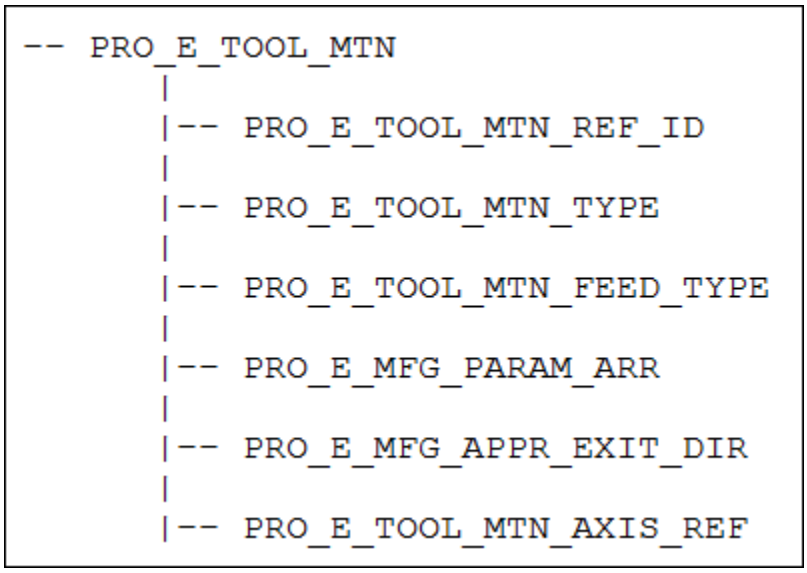
Element ID	Data Type	Description
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Species the type of the tool motion used for the Creo NC sequence. The valid value for this element is `PRO_TM_TYPE_HELICAL_APPROACH`. The value for this element is defined by `ProTmType`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are: • `PRO_TM_FEED_FREE`— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • `PRO_TM_FEED_CUT`— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • `PRO_TM_FEED_PLUNGE`— Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the

Element ID	Data Type	Description
		workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The following parameters are used to define the helical approach motion: • ENTRY_ANGLE • CLEAR_DIST • NORMAL_LEAD_STEP • LEAD_RADIUS The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of orientation for approach or exit tool motion. The direction of orientation is defined by the enumerated data type ProTmSideDir in ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection. **Note** This element is optional for 5-axis Creo NC sequences. By default Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences a user can select an alternative axis.

Tool Motion — Helical Exit

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnHelExit.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTNelement.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type. The valid value for this element is PRO_TM_TYPE_HELICAL_EXIT. The value for this element is defined by ProTmType.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_RETRACT—

Element ID	Data Type	Description
		Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_EXIT— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The following parameters are used to define the helical approach motion: • EXIT_ANGLE • PULLOUT_DIST • NORMAL_LEAD_STEP • LEAD_RADIUS The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Element ID	Data Type	Description
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of orientation for approach or exit tool motion. The direction of orientation is defined by the enumerated data type ProTmSideDir in ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection. **Note** This element is optional for 5-axis Creo NC sequences. By default Z-axis of the Creo NC sequence is used to define the constraint plane. For 5-axis Creo NC sequences a user can select an alternative axis.

Tool Motion — Go To Surface

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGotoSrf.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_TOOL_MTN_FEED_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
       |
       |-- PRO_E_TOOL_MTN_SRF_REF
```

The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_TYPE_GOTO_SURFACE. The value for this element is defined by ProTmType.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE—Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the workpiece. • PRO_TM_FEED_RETRACT—Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece. • PRO_TM_FEED_EXIT—Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_SRF_REF	PRO_VALUE_TYPE_ SELECTION	Mandatory element. Specifies the selection of a surface.

Tool Motion — Go To Axis

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGotoAxis.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_TOOL_MTN_FEED_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
       |
       |-- PRO_E_TOOL_MTN_GOTO_AXIS_REF
```

The following table lists the contents of PRO_E_TOOL_MTN element.

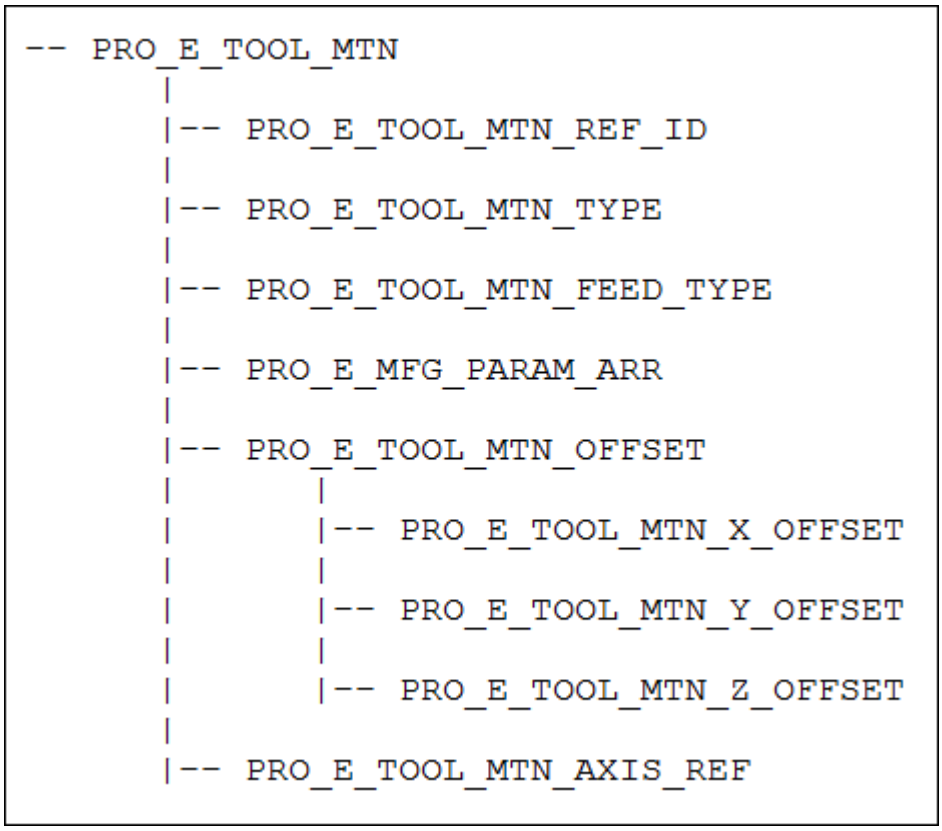
Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_ TYPE_GOTO_AXIS. The value for this element is defined by ProTmType.
PRO_E_TOOL_MTN_FEED_ TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type

Element ID	Data Type	Description
		ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE—Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the workpiece. • PRO_TM_FEED_RETRACT—Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece. • PRO_TM_FEED_EXIT—Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_GOTO_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the axis selection.

Tool Motion — Tangent Approach

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnTanAppr.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_TYPE_TANGENT_APPROACH.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are:

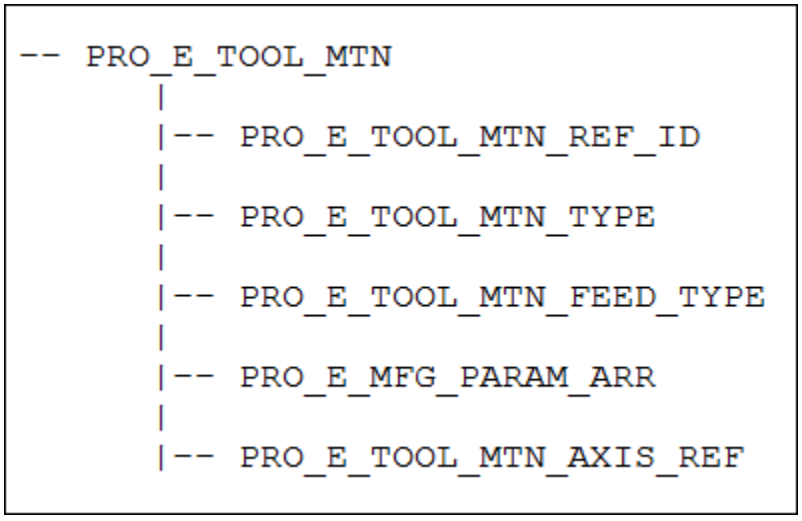
Element ID	Data Type	Description
		• PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The parameter APPROACH_DIST specifies the approach distance. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_OFFSET	Compound	Optional element. This compound element specifies the tool motion offset.
PRO_E_TOOL_MTN_X_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset value along the X-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Y_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset value along the Y-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset value along the Z-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Specifies the axis selection. **Note** • This element is optional for 5-axis Creo NC sequences • Z-axis of the Creo NC sequence is used to define the constraint plane. This is the default value. • For 5-axis Creo NC sequences you can select an alternative axis

Tool Motion — Tangent Exit

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnTanExit.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_TYPE_TANGENT_EXIT.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_EXIT— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The parameter APPROACH_DIST specifies the approach distance. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_ SELECTION	Specifies the axis selection. **Note** • This element is optional for 5-axis Creo NC sequences. • Z-axis of the Creo NC sequence is used to define the constraint plane. This is the default value. • For 5-axis Creo NC sequences you can select an alternative axis.

Tool Motion — Area and Groove Turning

The element `PRO_E_TOOL_MTN` is a member of `PRO_E_TOOL_MTN_ARR` array and is documented in the header file `ProMfgElemToolMtnAreaTurn.h` and `ProMfgElemToolMtnGrooveTurn.h` respectively, and is as shown in the following figure:

Element tree for PRO_E_TOOL_MTN element:

```
-- PRO_E_TOOL_MTN
   |
   |-- PRO_E_TOOL_MTN_REF_ID
   |
   |-- PRO_E_TOOL_MTN_TYPE
   |
   |-- PRO_E_MFG_PARAM_ARR
   |
   |-- PRO_E_TOOL_MTN_TURN_PROF
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_REF
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_S_VAL
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_E_VAL
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_S_REF
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_E_REF
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_S_ORIENT
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_PROF_E_ORIENT
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_DFLT_CORNER_TYPE
   |       |
   |       |-- PRO_E_TOOL_MTN_TURN_CORNER_ARR
   |       |
   |       |-- PRO_E_TURN_STK_ALLW_PROF_ARR
   |       |
   |       |-- PRO_E_TURN_STK_ALLW_ROUGH_ARR
   |
   |-- PRO_E_TOOL_MTN_TURN_STK_BND_REF
   |
   |-- PRO_E_TOOL_MTN_TURN_EXT
           |
           |-- PRO_E_TOOL_MTN_TURN_S_EXT
           |
           |-- PRO_E_TOOL_MTN_TURN_E_EXT
```

The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the Tool motion type. The valid value for this element is • PRO_TM_TYPE_AREA _TURNING—For area turning. • PRO_TM_TYPE_GROOVE _TURNING—For groove turning. . The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_TURN_ PROF	Compound	This compound element specifies the turning profile definition.
PRO_E_TOOL_MTN_TURN_ PROF_REF	PRO_VALUE_TYPE_ SELECTION	Mandatory element. Specifies the turning profile reference.
PRO_E_TOOL_MTN_TURN_ PROF_S_VAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the start point offset. This element can range from negative to positive values. The valid range values for this element are from -MAX_ DIM_VALUE to MAX_DIM_ VALUE.
PRO_E_TOOL_MTN_TURN_ PROF_E_VAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the end point offset. This element can range from negative to positive values. The valid range values for this element are from -MAX_ DIM_VALUE to MAX_DIM_ VALUE.
PRO_E_TOOL_MTN_TURN_ PROF_S_REF	PRO_VALUE_TYPE_ SELECTION	Optional element. Specifies the start point adjustment reference.
PRO_E_TOOL_MTN_TURN_	PRO_VALUE_TYPE_	Optional element. Specifies the

Element ID	Data Type	Description
PROF_E_REF	SELECTION	end point adjustment reference.
PRO_E_TOOL_MTN_TURN_ PROF_S_ORIENT	PRO_VALUE_TYPE_INT	Specifies the orientation of the axis when coordinate system is selected as the start point adjustment. The valid values for this element are: • PRO_TM_TURN_CSYS_X • PRO_TM_TURN_CSYS_Y • PRO_TM_TURN_CSYS_Z **Note** This element is mandatory if coordinate system is selected for cylindrical retract reference and is ignored in all other cases.
PRO_E_TOOL_MTN_TURN _PROF_E_ORIENT	PRO_VALUE_TYPE_INT	Specifies the orientation of the axis when coordinate system is selected as the ending point. The valid values for this element are: • PRO_TM_TURN_CSYS_X • PRO_TM_TURN_CSYS_Y • PRO_TM_TURN_CSYS_Z **Note** This element is mandatory if coordinate system is selected for cylindrical retract reference and is ignored in all other cases.
PRO_E_TOOL_MTN_TURN _DFLT_CORNER_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the default corner type. The valid values for this element are: • PRO_TM_TURN_CORNER_ TYPE_SHARP • PRO_TM_TURN_CORNER_ TYPE_FILLET • PRO_TM_TURN_CORNER_ TYPE_CHAMFER
PRO_E_TOOL_MTN_TURN_ CORNER_ARR	Array	Optional element. Specifies the corner conditions array.
PRO_E_TOOL_MTN_TURN_	Compound	This compound element defines

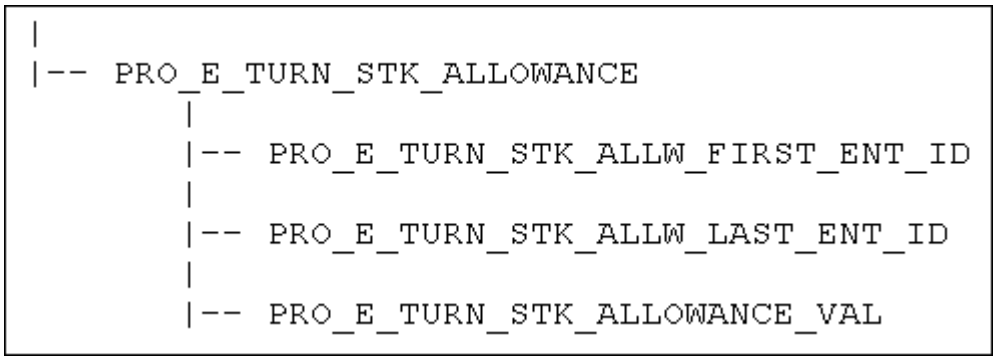
Element ID	Data Type	Description
CORNER		the elements related to the corner
PRO_E_TOOL_MTN_TURN_CORNER_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the corner type. The valid values for this element are: • PRO_TM_TURN_CORNER_TYPE_SHARP • PRO_TM_TURN_CORNER_TYPE_FILLET • PRO_TM_TURN_CORNER_TYPE_CHAMFER
PRO_E_TOOL_MTN_TURN_PREV_ENT_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the previous entity id.
PRO_E_TOOL_MTN_TURN_NEXT_ENT_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the next entity id.
PRO_E_TOOL_MTN_TURN_CORNER_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the fillet radius or chamfer dimension. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE **Note** This element is mandatory if corner type is PRO_TM_TURN_CORNER_TYPE_FILLET or PRO_TM_TURN_CORNER_TYPE_CHAMFER.
PRO_E_TURN_STK_ALLW_PROF_ARR	Array	Specifies an array for profile stock allowance.
PRO_E_TURN_STK_ALLW_ROUGH_ARR	Array	Specifies an array for rough stock allowance.
PRO_E_TURN_STK_ALLOWANCE	Compound	Specifies the compound element for stock allowance. For more information, refer to the section Specifying the Stock Allowance on page 1735.
PRO_E_TOOL_MTN_TURN_STK_BND_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies reference to the workpiece or stock boundary.
PRO_E_TOOL_MTN_TURN_EXT	Compound	Mandatory element. This compound element specifies extensions.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TURN_S_EXT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the start extension. The valid values for this element are: • EXT_POS_Z • EXT_NEG_Z • EXT_POS_Y • EXT_NEG_Y • EXT_NONE
PRO_E_TOOL_MTN_TURN_E_EXT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the end extension. The valid values for this element are: • EXT_POS_Z • EXT_NEG_Z • EXT_POS_Y • EXT_NEG_Y • EXT_NONE

Specifying the Stock Allowance

The element PRO_E_TURN_STK_ALLOWANCE is used to specify the stock allowance and is documented in the header files ProMfgElemToolMtnAreaTurn.h, ProMfgElemToolMtnGrooveTurn.h and ProMfgElemToolMtnProfTurn.h.

The following figure shows the element PRO_E_TURN_STK_ALLOWANCE:



The following table lists the contents of PRO_E_TURN_STK_ALLOWANCE element.

Element ID	Data Type	Description
PRO_E_TURN_STK_ALLOWANCE	Compound	Specifies the compound element for stock allowance.
PRO_E_TURN_STK_ALLW_	PRO_VALUE_TYPE_INT	Specifies the ID of the first entity

Element ID	Data Type	Description
FIRST_ENT_ID		of the turn profile segment with stock allowance.
PRO_E_TURN_STK_ALLW_ LAST_ENT_ID	PRO_VALUE_TYPE_INT	Specifies the ID of the last entity of the turn profile segment with stock allowance.
PRO_E_TURN_STK_ ALLOWANCE_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the value of the stock allowance.

Tool Motion — Profile Turning

The element `PRO_E_TOOL_MTN` is a member of `PRO_E_TOOL_MTN_ARR` array and is documented in the header file `ProMfgElemToolMtnProfTurn.h`, and is as shown in the following figure:

Element tree for PRO_E_TOOL_MTN element:

```
-- PRO_E_TOOL_MTN
    |
    |-- PRO_E_TOOL_MTN_REF_ID
    |
    |-- PRO_E_TOOL_MTN_TYPE
    |
    |-- PRO_E_MFG_PARAM_ARR
    |
    |-- PRO_E_TOOL_MTN_TURN_PROF
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_REF
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_S_VAL
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_E_VAL
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_S_REF
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_E_REF
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_S_ORIENT
            |
            |-- PRO E TOOL MTN TURN PROF E ORIENT
            |
            |-- PRO_E_TOOL_MTN_TURN_PROF_OFFSET_CUT
            |
            |-- PRO_E_TOOL_MTN_TURN_DFLT_CORNER_TYPE
            |
            |-- PRO_E_TOOL_MTN_TURN_CORNER_ARR
            |       |
            |       |-- PRO_E_TOOL_MTN_TURN_CORNER
            |               |
            |               |-- PRO_E_TOOL_MTN_TURN_CORNER_TYPE
            |               |
            |               |-- PRO_E_TOOL_MTN_TURN_PREV_ENT_ID
            |               |
            |               |-- PRO_E_TOOL_MTN_TURN_NEXT_ENT_ID
            |               |
            |               |-- PRO_E_TOOL_MTN_TURN_CORNER_VAL
            |
            |-- PRO_E_TURN_STK_ALLW_PROF_ARR
            |
            |-- PRO_E_TURN_STK_ALLW_ROUGH_ARR
```

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the Tool motion type. The valid value for this element is PRO_TM_TYPE_PROF_TURNING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC online help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_TURN_PROF	Compound	This compound element specifies the turning profile definition.
PRO_E_TOOL_MTN_TURN_PROF_REF	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the turning profile reference.
PRO_E_TOOL_MTN_TURN_PROF_S_VAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the start point offset. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_TURN_PROF_E_VAL	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the end point offset. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_TURN_PROF_S_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the start point adjustment reference.
PRO_E_TOOL_MTN_TURN_PROF_E_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the end point adjustment reference.
PRO_E_TOOL_MTN_TURN_PROF_S_ORIENT	PRO_VALUE_TYPE_INT	Specifies the orientation of the axis when coordinate system is selected as the start point adjustment. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_TM_TURN_CSYS_X • PRO_TM_TURN_CSYS_Y • PRO_TM_TURN_CSYS_Z **Note** This element is mandatory if coordinate system is selected for cylindrical retract reference and is ignored in all other cases.
PRO_E_TOOL_MTN_TURN_PROF_E_ORIENT	PRO_VALUE_TYPE_INT	Specifies the orientation of the axis when coordinate system is selected as the ending point. The valid values for this element are: • PRO_TM_TURN_CSYS_X • PRO_TM_TURN_CSYS_Y • PRO_TM_TURN_CSYS_Z **Note** This element is mandatory if coordinate system is selected for cylindrical retract reference and is ignored in all other cases.
PRO_E_TOOL_MTN_TURN_PROF_OFFSET_CUT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the offset from turn profile by the tool radius. The valid values for this element are: • PRO_B_TRUE • PRO_B_FALSE
PRO_E_TOOL_MTN_TURN_DFLT_CORNER_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the corner type. The valid values for this element are: • PRO_TM_TURN_CORNER_TYPE_SHARP • PRO_TM_TURN_CORNER_TYPE_FILLET • PRO_TM_TURN_CORNER_TYPE_CHAMFER
PRO_E_TOOL_MTN_TURN_CORNER_ARR	Array	Optional element. Specifies the corner conditions array.
PRO_E_TOOL_MTN_TURN_CORNER	Compound	This compound element defines the elements related to the corner.
PRO_E_TOOL_MTN_TURN_CORNER_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the corner type. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_TM_TURN_CORNER_TYPE_SHARP • PRO_TM_TURN_CORNER_TYPE_FILLET • PRO_TM_TURN_CORNER_TYPE_CHAMFER
PRO_E_TOOL_MTN_TURN_PREV_ENT_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the previous entity id.
PRO_E_TOOL_MTN_TURN_NEXT_ENT_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the next entity id.
PRO_E_TOOL_MTN_TURN_CORNER_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the fillet radius or chamfer dimension. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE **Note** This element is mandatory if corner type is PRO_TM_TURN_CORNER_TYPE_FILLET or PRO_TM_TURN_CORNER_TYPE_CHAMFER.
PRO_E_TURN_STK_ALLW_PROF_ARR	Array	Specifies an array for profile stock allowance.
PRO_E_TURN_STK_ALLW_ROUGH_ARR	Array	Specifies an array for rough stock allowance.
PRO_E_TURN_STK_ALLOWANCE	Compound	Specifies the compound element for stock allowance. For more information, refer to the section Specifying the Stock Allowance on page 1735.

Tool Motion — Curve Trajectory

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnCrvTraj.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
      |
      |-- PRO_E_TOOL_MTN_REF_ID
      |
      |-- PRO_E_TOOL_MTN_TYPE
      |
      |-- PRO_E_MFG_PARAM_ARR
      |
      |-- PRO_E_MFG_TRAJ_CRV
      |       |
      |       |-- PRO_E_STD_CURVE_COLLECTION_APPL
      |
      |-- PRO_E_NCD_CURVE_POINT
      |       |
      |       |-- PRO_E_NCD_CURVE_POINT_REF
      |       |
      |       |-- PRO_E_NCD_CURVE_POINT_OFFSET_TYPE
      |       |
      |       |-- PRO_E_NCD_CURVE_POINT_OFFSET
      |
      |-- PRO_E_MFG_HELICAL_CUT_OPT
      |
      |-- PRO_E_MFG_START_HEIGHT
      |
      |-- PRO_E_MFG_HEIGHT
      |
      |-- PRO_E_MFG_OFFSET
      |       |
      |       |-- PRO_E_MFG_OFFSET_CUT
      |       |
      |       |-- PRO_E_MFG_MAT_TO_RMV
      |       |
      |       |-- PRO_E_MFG_DRV_SRF_DIR
      |
      |-- PRO_E_CHECK_SURF_COLL
      |
      |-- PRO_E_MFG_AXIS_DEF_COMP
      |
      |-- PRO_E_MFG_TRAJ_CORNER_COND
             |
             |-- PRO_E_MFG_TRAJ_CORNER_DFLT_TYPE
             |
             |-- PRO_E_MFG_TRAJ_CORNER_ARR
                    |
                    |-- PRO_E_MFG_TRAJ_CORNER
                           |
                           |-- PRO_E_MFG_TRAJ_CORNER_TYPE
                           |
                           |-- PRO_E_MFG_TRAJ_CORNER_PREV_ID
                           |
                           |-- PRO_E_MFG_TRAJ_CORNER_NEXT_ID
                           |
                           |-- PRO_E_MFG_TRAJ_CORNER_VAL
```

The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is PRO_TM_TYPE_CURVE_TRAJECTORY.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_TRAJ_CRV	Compound	Mandatory element. Specifies the machining curves compound definition.
PRO_E_STD_CURVE_COLLECTION_APPL	Curve Collection	Mandatory element. Specifies the curve collection.
PRO_E_NCD_CURVE_POINT	Compound	Optional element. Specifies the compound Start Point definition. **Note** This element is applicable for closed loops only.
PRO_E_NCD_CURVE_POINT_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the start point vertex definition to offset from.
PRO_E_NCD_CURVE_POINT_OFFSET_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the start point offset type definition. The valid values for this element are: • PRO_CURVE_POINT_OFFSET_TYPE_RATIO— Specifies the offset by parameter. • PRO_CURVE_POINT_OFFSET_TYPE_REAL— Specifies the offset by length.
PRO_E_NCD_CURVE_POINT_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the start point offset definition.
PRO_E_MFG_HELICAL_CUT_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the helical cut option. The valid values for this element are:

Element ID	Data Type	Description
		• PRO_B_TRUE—Specifies that helical option and parameters will be applied. • PRO_B_FALSE—Specifies that helical option and parameters will not be applied.
PRO_E_MFG_START_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of start height surface.
PRO_E_MFG_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the multiple selections of height surfaces.
PRO_E_MFG_OFFSET	Compound	Optional element. Specifies the offset compound definition.
PRO_E_MFG_OFFSET_CUT	PRO_VALUE_TYPE_INT	Optional element. Specifies the offset cut. The valid values for this element are: • PRO_B_TRUE—Tool offset will be applied. • PRO_B_FALSE—Tool offset will not be applied.
PRO_E_MFG_MAT_TO_RMV	PRO_VALUE_TYPE_INT	Optional element. Specifies the material side. The values for this element are defined by the enumerated value ProMaterialRmvSide in the header file ProMfgOptions.h. The valid values for this element are: • PRO_MAT_RMV_LEFT —Specifies a cut on left from the curve. • PRO_MAT_RMV_RIGHT —Specifies a cut on right from the curve.
PRO_E_MFG_DRV_SRF_DIR	PRO_VALUE_TYPE_INT	Optional element. Specifies the flip drive surface direction. The valid values for this element are: • PRO_B_FALSE—The default direction on the drive surface will be used. • PRO_B_TRUE—The opposite direction on the drive surface will be used.
PRO_E_CHECK_SURF_COLL	Compound	Specifies the check surfaces compound definition. The element tree for the Checking Surfaces is defined in the header file ProMfgElemCheckSurf.h. For more information, refer to the section Checking Surfaces on page 1687 for more information on the element tree.

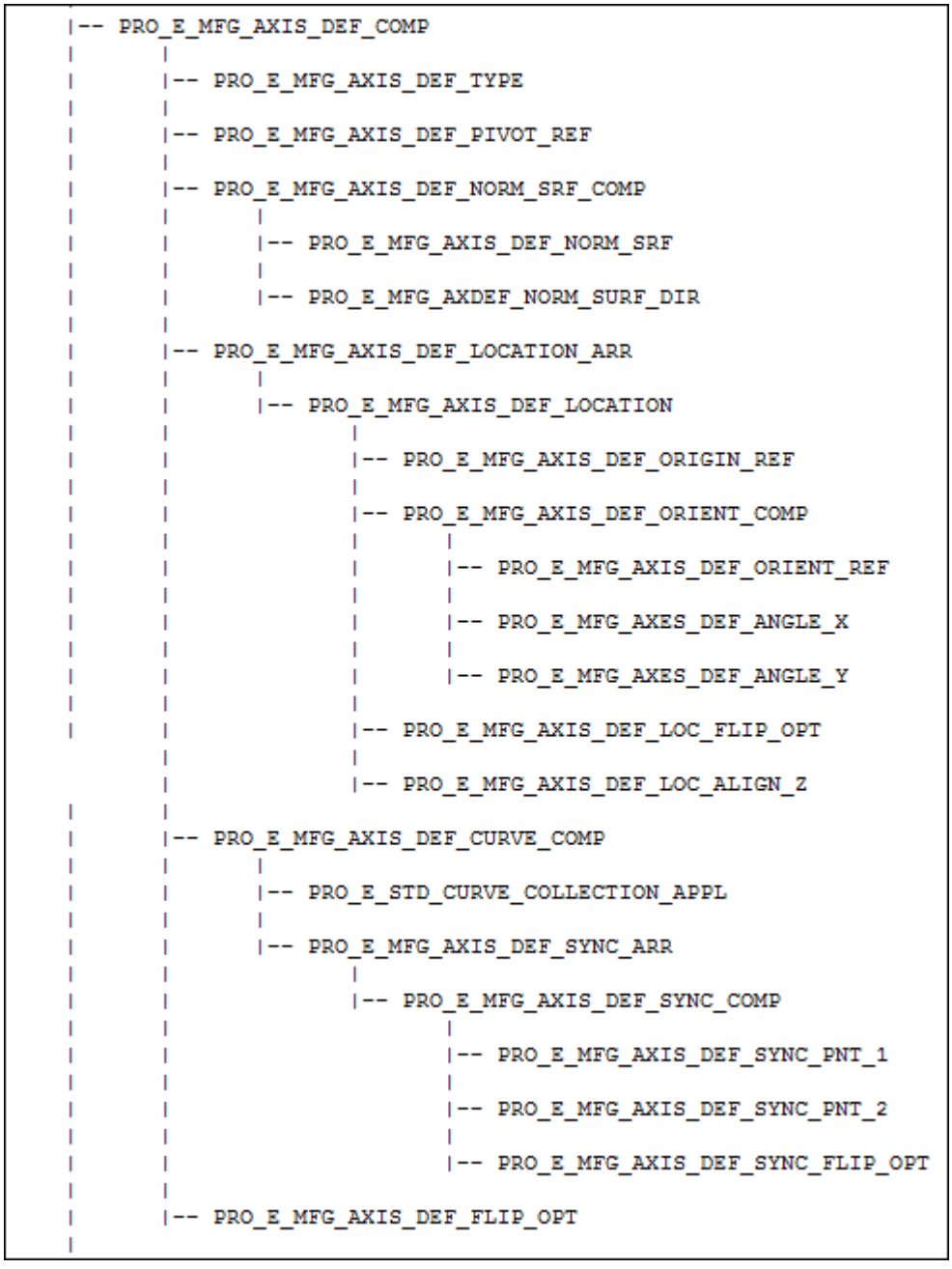
Element ID	Data Type	Description
PRO_E_MFG_AXIS_DEF_COMP	Compound	Optional element. Specifies the compound element for the axis definition.
PRO_E_MFG_TRAJ_CORNER_COND	Compound	Optional element. Specifies the compound element for the corner condition.
PRO_E_MFG_TRAJ_CORNER_DFLT_TYPE	PRO_VALUE_TYPE_INT	Specifies the default corner type. The valid values for this element are defined in the enumerated type ProTmTrajCornerType and are as follows: • PRO_TM_TRAJ_CORNER_TYPE_SHARP • PRO_TM_TRAJ_CORNER_TYPE_FILLET • PRO_TM_TRAJ_CORNER_TYPE_CHAMFER • PRO_TM_TRAJ_CORNER_TYPE_LOOP • PRO_TM_TRAJ_CORNER_TYPE_STRAIGHT
PRO_E_MFG_TRAJ_CORNER_ARR	Array	Optional element. Specifies an array for the corner condition.
PRO_E_MFG_TRAJ_CORNER	Compound	Optional element. Specifies the corner condition item.
PRO_E_MFG_TRAJ_CORNER_TYPE	PRO_VALUE_TYPE_INT	Specifies the default corner type. The valid values for this element are defined in the enumerated type ProTmTrajCornerType and are as follows: • PRO_TM_TRAJ_CORNER_TYPE_SHARP • PRO_TM_TRAJ_CORNER_TYPE_FILLET • PRO_TM_TRAJ_CORNER_TYPE_CHAMFER • PRO_TM_TRAJ_CORNER_TYPE_LOOP • PRO_TM_TRAJ_CORNER_TYPE_STRAIGHT
PRO_E_MFG_TRAJ_CORNER_PREV_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the previous Id for the corner.

Element ID	Data Type	Description
PRO_E_MFG_TRAJ_CORNER_NEXT_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the next Id for the corner.
PRO_E_MFG_TRAJ_CORNER_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the fillet radius or the chamfer dimension. The valid range for this element is from 0 to MAX_DIM_VALUE. **Note** This element is mandatory if the corner type is set to PRO_TM_TRAJ_CORNER_TYPE_FILLET or PRO_TM_TRAJ_CORNER_TYPE_CHAMFER.

Element Tree for PRO_E_MFG_AXIS_DEF_COMP

The element tree for `PRO_E_MFG_AXIS_DEF_COMP` is as shown in the figure below:

Element tree for PRO_E_MFG_AXIS_DEF_COMP element



The following table lists the contents of PRO_E_MFG_AXIS_DEF_COMP element.

Element ID	Data Type	Description
PRO_E_MFG_AXIS_DEF_COMP	Compound	Optional element. Specifies the compound element for the axis definition.
PRO_E_MFG_AXIS_DEF_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the axis definition. The valid values for this element are defined in the enumerated type ProAxisDefType and are as follows: • PRO_AXIS_DEF_TYPE_UNDEF • PRO_AXIS_DEF_BY_PIVOT_REF • PRO_AXIS_DEF_BY_LOCATIONS • PRO_AXIS_DEF_BY_TWO_CONTOURS • PRO_AXIS_DEF_BY_NORM_SURF
PRO_E_MFG_AXIS_DEF_PIVOT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the single reference. You can select either a point or an axis. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_PIVOT_REF. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_NORM_SRF_COMP	Compound	Specifies the normal surface

Element ID	Data Type	Description
		compound element. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_NORM_SURF. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_NORM_SRF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of multiple references. You can select surface, quilt or feature. **Note** The quilt or feature must represent a Mill surface, if selected. This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_NORM_SURF. This element is ignored in all other cases.
PRO_E_MFG_AXDEF_NORM_SURF_DIR	PRO_VALUE_TYPE_INT	Specifies the normal surface direction. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction opposite to the normal to the surface is selected for the element PRO_E_MFG_AXIS_DEF_NORM_SRF. • PRO_B_FALSE—Specifies that same direction as the normal to the surface is selected for the element PRO_E_MFG_AXIS_

Element ID	Data Type	Description
		DEF_NORM_SRF. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_NORM_SURF. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_LOCATION_ARR	Array	Specifies an array of locations. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_LOCATIONS. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_LOCATION	Compound	Mandatory element. Specifies the compound element for the location axis definition.
PRO_E_MFG_AXIS_DEF_ORIGIN_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the single reference. You can either select a point on a curve or an edge.
PRO_E_MFG_AXIS_DEF_ORIENT_COMP	Compound	Mandatory element. Specifies the orientation compound element.
PRO_E_MFG_AXIS_DEF_ORIENT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the single reference selection. You can select either a point or an axis. **Note** This element is mandatory, if the elements PRO_E_MFG_AXES_DEF_ANGLE_X and PRO_E_MFG_AXES_DEF_ANGLE_Y are not defined.
PRO_E_MFG_AXES_DEF_ANGLE_X	PRO_VALUE_TYPE_DOUBLE	Specifies the lead angle. The valid range for this element is from —90

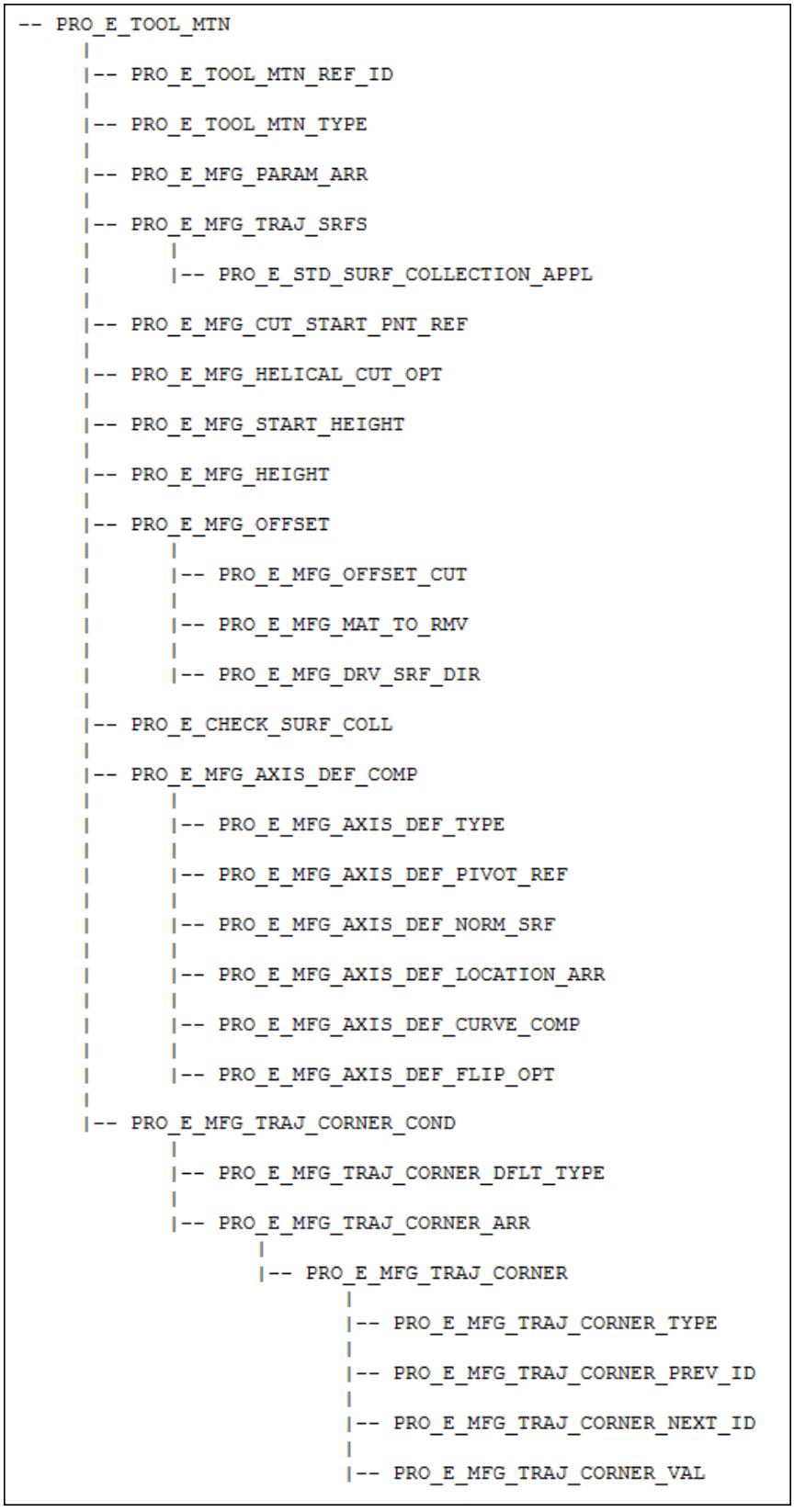
Element ID	Data Type	Description
		to +90. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_ORIENT_REF is not defined.
PRO_E_MFG_AXES_DEF_ANGLE_Y	PRO_VALUE_TYPE_DOUBLE	Specifies the tilt angle. The valid range for this element is from —90 to +90. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_ORIENT_REF is not defined.
PRO_E_MFG_AXIS_DEF_LOC_FLIP_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the flip direction at a location. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction will be reversed. • PRO_B_FALSE—Specifies that the direction will remain the same.
PRO_E_MFG_AXIS_DEF_LOC_ALIGN_Z	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the alignment of the tool axis with the Z axis of the step coordinate system. Specify the value PRO_B_TRUE to this element.
PRO_E_MFG_AXIS_DEF_CURVE_COMP	Compound	Specifies the compound element for the pivot curve. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_TWO_CONTOURS. This element is ignored in all other cases.
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Mandatory element. Specifies a general compound element for chain collection.
PRO_E_MFG_AXIS_DEF_	Array	Optional element. Specifies the

Element ID	Data Type	Description
SYNC_ARR		synchronization array.
PRO_E_MFG_AXIS_DEF_ SYNC_COMP	Compound	Optional element. Specifies the synchronization compound element.
PRO_E_MFG_AXIS_DEF_ SYNC_PNT_1	PRO_VALUE_TYPE_ SELECTION	Mandatory element. Specifies the single reference selection. Select a point on the trajectory curve.
PRO_E_MFG_AXIS_DEF_ SYNC_PNT_2	PRO_VALUE_TYPE_ SELECTION	Mandatory element. Specifies the single reference selection. Select a point on the pivot curve.
PRO_E_MFG_AXIS_DEF_ SYNC_FLIP_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the flip direction.The valid values for this element are: • PRO_B_TRUE—Specifies that the tool motion is flipped in the reverse direction. • PRO_B_FALSE—Specifies that the direction will remain the same.
PRO_E_MFG_AXIS_DEF_ FLIP_OPT	PRO_VALUE_TYPE_INT	Specifies the flip direction. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction will be reversed. • PRO_B_FALSE—Specifies that the direction will remain the same.

Tool Motion — Surface Trajectory

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_ MTN_ARR array and is documented in the header file ProMfgElemToolMtnSrfTraj.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type. The valid value for this element is PRO_TM_TYPE_SURF_TRAJECTORY.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_TRAJ_SRFS	Compound	Specifies the drive surfaces compound definition.
PRO_E_STD_SURF_COLLECTION_APPL	Surface Collection	Mandatory element. Specifies the drive surfaces collection.
PRO_E_MFG_CUT_START_PNT_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of a datum point or a point on the bottom edges of the machining surfaces. It allows the machining to start at the location which is nearest to the selected point. **Note** This element is applicable only when the machining surfaces form a closed loop.
PRO_E_MFG_HELICAL_CUT_OPT	PRO_VALUE_TYPE_INT	Optional element. Specifies the helical cut option. The valid values for this element are: • PRO_B_TRUE—Specifies that helical option and parameters will be applied. • PRO_B_FALSE—Specifies that helical option and parameters will not be applied.
PRO_E_MFG_START_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of start height surface.
PRO_E_MFG_HEIGHT	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the multiple selections of height surfaces.
PRO_E_MFG_OFFSET	Compound	Optional element. Specifies the offset compound definition.
PRO_E_MFG_OFFSET_CUT	PRO_VALUE_TYPE_INT	Optional element. Specifies the

Element ID	Data Type	Description
		offset cut. The valid values for this element are: • PRO_B_TRUE—Tool offset will be applied. • PRO_B_FALSE—Tool offset will not be applied.
PRO_E_MFG_MAT_TO_RMV	PRO_VALUE_TYPE_INT	Optional element. Specifies the material side. The valid values for this element are: • PRO_MFG_DIR_SAME—Default side will be used. • PRO_MFG_DIR_OPPOSITE—The default side will be flipped.
PRO_E_MFG_DRV_SRF_DIR	PRO_VALUE_TYPE_INT	Optional element. Specifies the flip drive Surface direction. The valid values for this element are: • PRO_B_FALSE—The default direction on the drive surface will be used. • PRO_B_TRUE—The opposite direction on the drive surface will be used.
PRO_E_CHECK_SURF_COLL	Compound	Optional element. Specifies the check surfaces compound definition. The element tree for the Checking Surfaces is defined in the header file ProMfgElemCheckSurf.h. For more information, refer to the section Checking Surfaces on page 1687 for more information on the element tree.
PRO_E_MFG_AXIS_DEF_COMP	Compound	Optional element. Specifies the compound element for the axis definition.
PRO_E_MFG_AXIS_DEF_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the axis definition. The valid values for this element are defined in the enumerated type ProAxisDefType and are as follows: • PRO_AXIS_DEF_TYPE_UNDEF • PRO_AXIS_DEF_BY_PIVOT_REF • PRO_AXIS_DEF_BY_LOCATIONS • PRO_AXIS_DEF_BY_TWO_CONTOURS • PRO_AXIS_DEF_BY_

Element ID	Data Type	Description
		NORM_SURF
PRO_E_MFG_AXIS_DEF_PIVOT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the single reference. You can select either a point or an axis. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_PIVOT_REF. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_NORM_SRF	Compound	Specifies the normal surface compound element. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_NORM_SURF. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_LOCATION_ARR	Array	Specifies an array of locations. **Note** This element is mandatory, only if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_LOCATIONS. This element is ignored in all other cases.
PRO_E_MFG_AXIS_DEF_LOCATION	Compound	Mandatory element. Specifies the compound element for the location axis definition.
PRO_E_MFG_AXIS_DEF_ORIGIN_REF	PRO_VALUE_TYPE_SELECTION	Specifies the selection of the single reference. You can either select a point on a curve or an edge.
PRO_E_MFG_AXIS_DEF_ORIENT_COMP	Compound	Mandatory element. Specifies the orientation compound element.
PRO_E_MFG_AXIS_DEF_ORIENT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the single reference selection. You can select either a

Element ID	Data Type	Description
		point or an axis. **Note** This element is mandatory, if the elements PRO_E_MFG_AXES_DEF_ANGLE_X and PRO_E_MFG_AXES_DEF_ANGLE_Y are not defined.
PRO_E_MFG_AXES_DEF_ANGLE_X	PRO_VALUE_TYPE_DOUBLE	Specifies the lead angle. The valid range for this element is from —90 to +90. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_ORIENT_REF is not defined.
PRO_E_MFG_AXES_DEF_ANGLE_Y	PRO_VALUE_TYPE_DOUBLE	Specifies the tilt angle. The valid range for this element is from —90 to +90. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_ORIENT_REF is not defined.
PRO_E_MFG_AXIS_DEF_LOC_FLIP_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the flip direction at a location. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction will be reversed. • PRO_B_FALSE—Specifies that the direction will remain the same.
PRO_E_MFG_AXIS_DEF_CURVE_COMP	Compound	Specifies the compound element for the pivot curve. **Note** This element is mandatory, if the element PRO_E_MFG_AXIS_DEF_TYPE is set to the value PRO_AXIS_DEF_BY_TWO_CONTOURS. This element is ignored in all other cases.

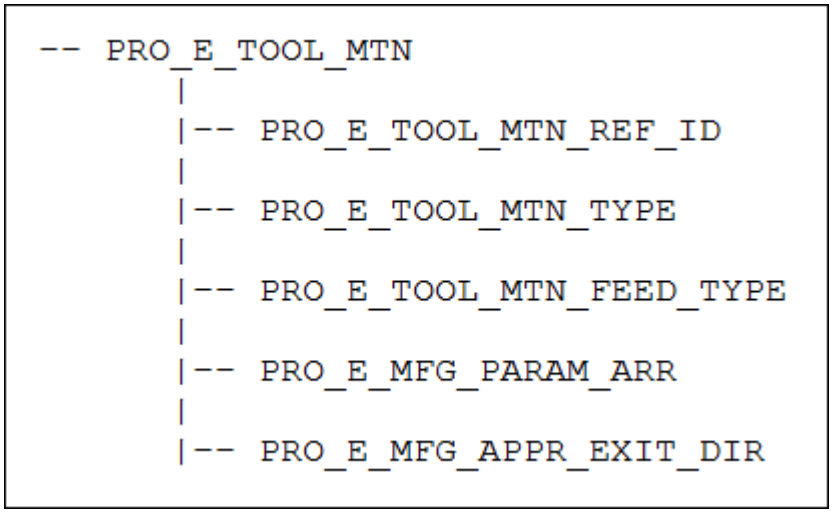
Element ID	Data Type	Description
PRO_E_STD_CURVE_COLLECTION_APPL	Chain Collection	Mandatory element. Specifies a general compound element for chain collection.
PRO_E_MFG_AXIS_DEF_SYNC_ARR	Array	Optional element. Specifies the synchronization array.
PRO_E_MFG_AXIS_DEF_SYNC_COMP	Compound	Optional element. Specifies the synchronization compound element.
PRO_E_MFG_AXIS_DEF_SYNC_PNT_1	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the single reference selection. Select a point on the trajectory curve.
PRO_E_MFG_AXIS_DEF_SYNC_PNT_2	PRO_VALUE_TYPE_SELECTION	Mandatory element. Specifies the single reference selection. Select a point on the pivot curve.
PRO_E_MFG_AXIS_DEF_FLIP_OPT	PRO_VALUE_TYPE_INT	Specifies the flip direction. The valid values for this element are: • PRO_B_TRUE—Specifies that the direction will be reversed. • PRO_B_FALSE—Specifies that the direction will remain the same.
PRO_E_MFG_TRAJ_CORNER_COND	Compound	Optional element. Specifies the compound element for the corner condition.
PRO_E_MFG_TRAJ_CORNER_DFLT_TYPE	PRO_VALUE_TYPE_INT	Specifies the default corner type. The valid values for this element are defined in the enumerated type ProTmTrajCornerType and are as follows: • PRO_TM_TRAJ_CORNER_TYPE_SHARP • PRO_TM_TRAJ_CORNER_TYPE_FILLET • PRO_TM_TRAJ_CORNER_TYPE_CHAMFER • PRO_TM_TRAJ_CORNER_TYPE_LOOP • PRO_TM_TRAJ_CORNER_TYPE_STRAIGHT
PRO_E_MFG_TRAJ_CORNER_ARR	Array	Optional element. Specifies an array for the corner condition.
PRO_E_MFG_TRAJ_CORNER	Compound	Optional element. Specifies the corner condition item.
PRO_E_MFG_TRAJ_CORNER_TYPE	PRO_VALUE_TYPE_INT	Specifies the default corner type. The valid values for this element are defined in the enumerated type ProTmTrajCornerType and are as follows:

Element ID	Data Type	Description
		• PRO_TM_TRAJ_CORNER_TYPE_SHARP • PRO_TM_TRAJ_CORNER_TYPE_FILLET • PRO_TM_TRAJ_CORNER_TYPE_CHAMFER • PRO_TM_TRAJ_CORNER_TYPE_LOOP • PRO_TM_TRAJ_CORNER_TYPE_STRAIGHT
PRO_E_MFG_TRAJ_CORNER_PREV_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the previous Id for the corner.
PRO_E_MFG_TRAJ_CORNER_NEXT_ID	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the next Id for the corner.
PRO_E_MFG_TRAJ_CORNER_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the fillet radius or the chamfer dimension. The valid range for this element is from 0 to MAX_DIM_VALUE. **Note** This element is mandatory if the corner type is set to PRO_TM_TRAJ_CORNER_TYPE_FILLET or PRO_TM_TRAJ_CORNER_TYPE_CHAMFER.

Tool Motion — Ramp Approach

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnRampAppr.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

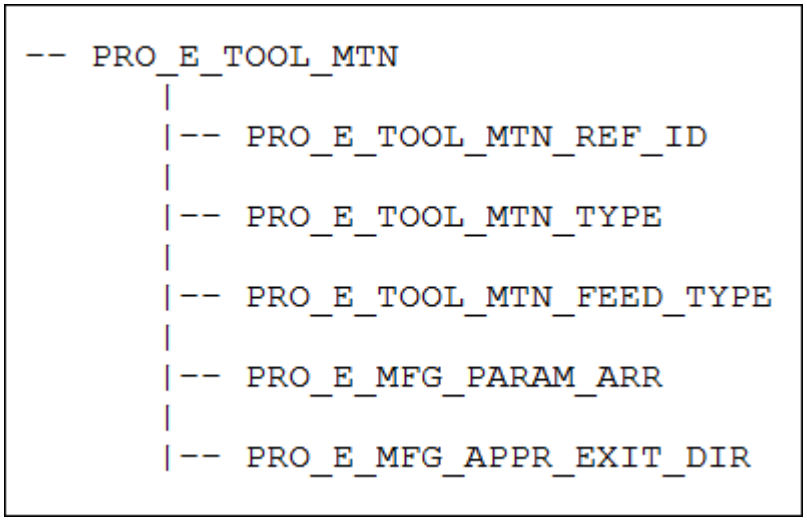
Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type. The valid value for this element is PRO_TM_TYPE_RAMP_APPROACH.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE—Specifies a plunge feed type. Plunge feed specifies the rate at which the tool approaches and plunges into the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the workpiece.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. Elements that define the ramp exit motion are: • EXIT_DIST • EXIT_ANGLE • RAMP_ANGLE See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction for the ramp approach type of tool exit. The direction is defined by the enumerated data type ProTmSideDir in ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE

Tool Motion — Ramp Exit

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnRampExit.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of `PRO_E_TOOL_MTN` element.

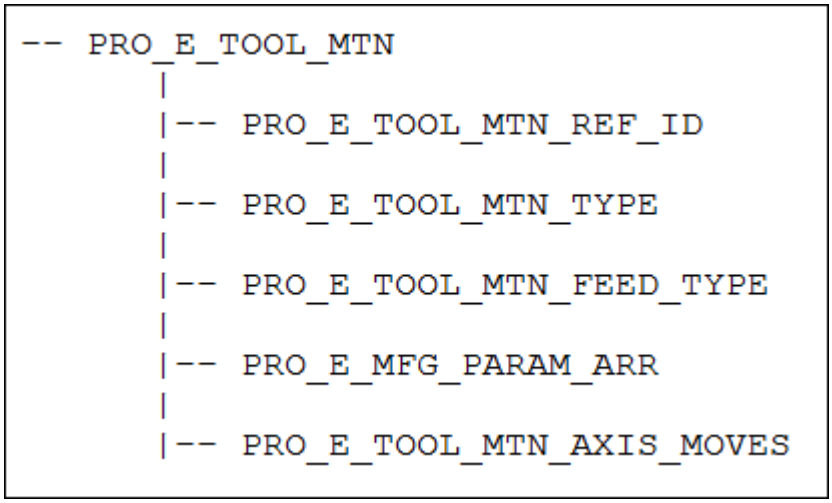
Element ID	Data Type	Description
`PRO_E_TOOL_MTN_TYPE`	`PRO_VALUE_TYPE_INT`	Mandatory element. Specifies the tool motion type. The valid value for this element is `PRO_TM_TYPE_RAMP_EXIT`.
`PRO_E_TOOL_MTN_FEED_TYPE`	`PRO_VALUE_TYPE_INT`	Optional element. Specifies the type of feed for the tool motion using the enumerated data type `ProToolMtnFeedType`. The valid value for this element are: • `PRO_TM_FEED_FREE`— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • `PRO_TM_FEED_CUT`— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • `PRO_TM_FEED_RETRACT`— Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • `PRO_TM_FEED_EXIT`— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.

Element ID	Data Type	Description
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. Elements that define the ramp exit motion are: • EXIT_ANGLE • TANGENT_LEAD_STEP • NORMAL_LEAD_STEP • LEAD_RADIUS See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_APPR_EXIT_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction for the ramp approach type of tool exit. The direction is defined by the enumerated data type ProTmSideDirin ProMfgOptions.h. The valid values for this element are: • PRO_TM_DIR_RIGHT_SIDE • PRO_TM_DIR_LEFT_SIDE

Tool Motion — Connect

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnConnect.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type used for the Creo NC sequence. The valid value for this element is PRO_TM_TYPE_CONNECT.
PRO_E_TOOL_MTN_FEED_TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE—Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT—Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_RETRACT—Specifies a retract feed type. Retract feed specifies the rate at which the tool moves away from the workpiece. • PRO_TM_FEED_APPROACH—Specifies an approach feed type. Approach feed specifies the rate at which the tool approaches the

Element ID	Data Type	Description
		workpiece. • PRO_TM_FEED_EXIT— Specifies an exit feed type. Exit feed specifies the rate at which the tool leaves the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_AXIS_MOVES	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the axis move attributes.

Tool Motion — Profile Mill Cut

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array is documented in the header file ProMfgElemToolMtnProfileMillCut.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
      |
      |-- PRO_E_TOOL_MTN_REF_ID
      |
      |-- PRO_E_TOOL_MTN_TYPE
      |
      |-- PRO_E_TOOL_MTN_PROFILE_TYPE
      |
      |-- PRO_E_MFG_PARAM_ARR
      |
      |-- PRO_E_MFG_CMP_APPROACH_EXIT
      |
      |-- PRO_E_MFG_START_HEIGHT
      |
      |-- PRO_E_MFG_HEIGHT
```

The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type. The valid value for this element is PRO_TM_TYPE_PROFILE_MILL_CUT.
PRO_E_TOOL_MTN_PROFILE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the tool motion profile. The values for this element are defined by Pro_MillProfCutType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_MFG_CMP_APPROACH_EXIT	Compound	Optional element. Specifies approach and exit compound definition. For more information, refer to the section Approach and Exit on page 1689for more information on the element tree.

Element ID	Data Type	Description
PRO_E_MFG_START_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the starting height and selection of a horizontal surface. **Note** This element is mandatory if the element PRO_E_TOOL_MTN_PROFILE_TYPE is set to PRO_E_MILL_CUT_FROM_TO. This element is ignored otherwise.
PRO_E_MFG_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the height and enables single surface selection. **Note** This element is mandatory if the element PRO_E_TOOL_MTN_PROFILE_TYPE is set to PRO_E_MILL_CUT_FROM_TO,PRO_E_MILL_CUT_UPTO and PRO_E_MILL_CUT_ONE_SLICE. This element is ignored otherwise.

Tool Motion — Auto Cut

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnAutoCut.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
```

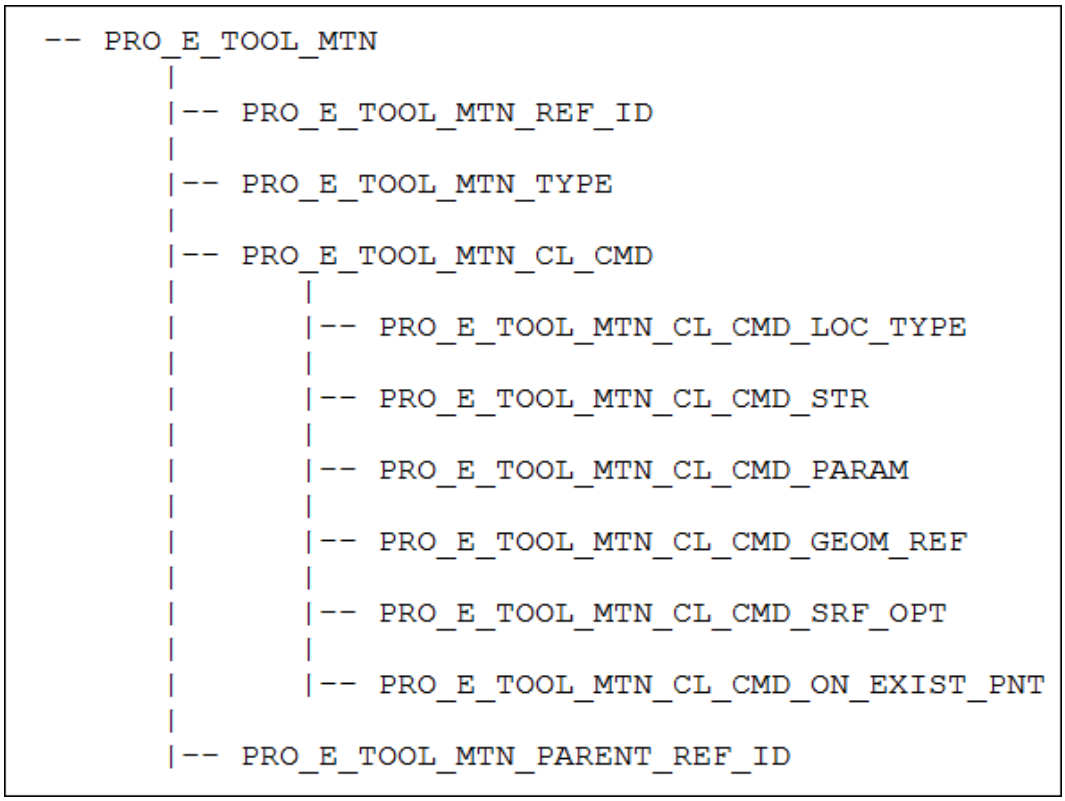
The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type. The valid value for this element is PRO_TM_TYPE_AUTOMATIC_CUT.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — CL Command

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnClCmd.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type. The valid value for this element is PRO_TM_TYPE_CL_COMMAND.
PRO_E_TOOL_MTN_CL_CMD	Compound	This compound element defines elements related to CL command.
PRO_E_TOOL_MTN_CL_CMD_LOC_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the CL command location types. The valid values for this element are: • PRO_CL_CMD_LOC_TYPE_SEL—Specifies that the CL command position is on tool path. **Note** The value for the element PRO_E_TOOL_MTN_CL_CMD_PARAM must be assigned. • PRO_CL_CMD_LOC_TYPE_BEGIN—Specifies that the CL command position is at the beginning of tool path. • PRO_CL_CMD_LOC_TYPE_CURR—Specifies that the CL command position is at the last point of the previous motion. • PRO_CL_CMD_LOC_TYPE_DTM_PNT—Specifies that the CL command position is at the projection of given point on tool path. **Note** The element NCL_CMD_LOC_TYPE_DTM_PNT must be defined. • PRO_CL_CMD_LOC_TYPE_ON_SURFACE— Specifies that the CL command positions are at intersection s of given

Element ID	Data Type	Description
		surface and tool path. **Note** The element PRO_CL_CMD_LOC_TYPE_ON_SURFACE must be defined
PRO_E_TOOL_MTN_CL_CMD_STR	PRO_VALUE_TYPE_WSTRING	Mandatory element. Specifies a user specified string.
PRO_E_TOOL_MTN_CL_CMD_PARAM	PRO_VALUE_TYPE_DOUBLE	Specifies the range for the CL command parameter. The valid values for this element are 0 or 1. **Note** This element is mandatory for PRO_CL_CMD_LOC_TYPE_SEL location type.
PRO_E_TOOL_MTN_CL_CMD_GEOM_REF	PRO_VALUE_TYPE_SELECTION	Specifies the reference for a datum point or a surface. **Note** This element is mandatory for NCL_CMD_LOC_TYPE_DTM_PNT and PRO_CL_CMD_LOC_TYPE_ON_SURFACE location types.
PRO_E_TOOL_MTN_CL_CMD_SRF_OPT	PRO_VALUE_TYPE_INT	Specifies the CL command surface options. The valid values for this element are: • PRO_CL_COMMAND_ON_FIRST_PASS—Specifies that only the first pass is considered for surface/tool path intersection . • PRO_CL_COMMAND_ON_LAST_PASS—Specifies that only the last pass is considered for surface/tool path intersection . • PRO_CL_COMMAND_ON_ALL_PASSES—Specifies that all passes are considered for

Element ID	Data Type	Description
		surface/tool path intersection . **Note** This element is mandatory for the element NCL_CMD_LOC_TYPE_ON_SURFACE.
PRO_E_TOOL_MTN_CL_CMD_ON_EXIST_PNT	PRO_VALUE_TYPE_INT	The valid values for this element are: • PRO_B_TRUE—It places the CL command at the nearest existing point of the tool path; • PRO_B_FALSE—Adds new point to the tool path. **Note** This element is mandatory for PRO_CL_CMD_LOC_TYPE_DTM_PNT.
PRO_E_TOOL_MTN_PARENT_REF_ID	PRO_VALUE_TYPE_INT	Specifies the tool motion id of the tool motion referred by the cl command tool motion. **Note** This element is mandatory for PRO_CL_CMD_LOC_TYPE_SEL.

Tool Motion — Follow Cut

The compound element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnFollowCut.h. The Follow Cut tool Motion follows the following types of tool motions in an array:

- PRO_TM_TYPE_CURVE_TRAJECTORY
- PRO_TM_TYPE_AREA_TURNING
- PRO_TM_TYPE_GROOVE_TURNING
- PRO_TM_TYPE_PROF_TURNING
- PRO_TM_TYPE_SURF_TRAJECTORY
- PRO_TM_TYPE_EDGE_TRAJECTORY

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
       |
       |-- PRO_E_TOOL_MTN_PARENT_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TRIM_VAL_OPT
       |
       |-- PRO_E_TOOL_MTN_TRIM_START_VAL
       |
       |-- PRO_E_TOOL_MTN_TRIM_END_VAL
       |
       |-- PRO_E_TOOL_MTN_TRIM_FLIP_DIR
```

The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type. The valid value for this element is PRO_TM_TYPE_TRIM. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_TRIM_VAL_OPT	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the value options. The valid values for this element are: • PRO_TM_TRIM_VAL_PARAM—The input value is a ratio parameter. • PRO_TM_TRIM_VAL_

Element ID	Data Type	Description
		DIST—The input value is a distance.
PRO_E_TOOL_MTN_TRIM_START_VAL	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the start parameter on tool path. The range specifies by this element is from [0., 1.].
PRO_E_TOOL_MTN_TRIM_END_VAL	PRO_VALUE_TYPE_DOUBLE	Mandatory element. Specifies the end parameter on tool path. The range specifies by this element is from [0., 1.]. **Note** The value for the element PRO_E_TOOL_MTN_TRIM_END_VAL must be greater than PRO_E_TOOL_MTN_TRIM_START_VAL
PRO_E_TOOL_MTN_TRIM_FLIP_DIR	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the direction of the tool motion. The valid values for this element are: • PRO_B_TRUE—Specifies reversed direction. • PRO_B_FALSE—Specifies same direction.

Tool Motion — Plunge

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnPlunge.h, and is as shown in the following figure:

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_TOOL_MTN_FEED_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
       |
       |-- PRO_E_TOOL_MTN_OFFSET
       |       |
       |       |-- PRO_E_TOOL_MTN_X_OFFSET
       |       |
       |       |-- PRO_E_TOOL_MTN_Y_OFFSET
       |       |
       |       |-- PRO_E_TOOL_MTN_Z_OFFSET
       |
       |-- PRO_E_TOOL_MTN_AXIS_REF
```

The following table describes the elements in the element tree for the approach along tool axis feature.

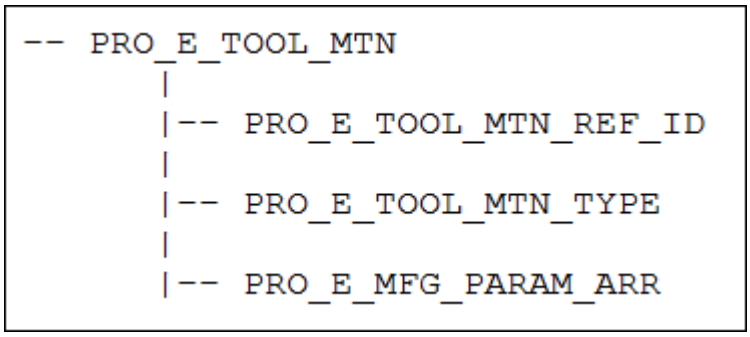
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Species the tool motion type . The valid value for this element is Tool Motion — Plunge on page 1772.
PRO_E_TOOL_MTN_FEED_ TYPE	PRO_VALUE_TYPE_INT	Optional element. Specifies the type of feed for the tool motion using the enumerated data type ProToolMtnFeedType. The valid value for this element are: • PRO_TM_FEED_FREE— Specifies a free feed type. Free feed specifies the rate at which the tool moves in the transverse motion, that is, the non-cutting motion. • PRO_TM_FEED_CUT— Specifies a cut feed type. Cut feed specifies the rate at which the tool moves into the workpiece. • PRO_TM_FEED_PLUNGE— Specifies a plunge feed type. Plunge feed specifies the rate

		at which the tool approaches and plunges into the workpiece.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.
PRO_E_TOOL_MTN_OFFSET	Compound	Optional element. This compound element specifies the offset value.
PRO_E_TOOL_MTN_X_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along X-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Y_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Y-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE.
PRO_E_TOOL_MTN_Z_OFFSET	PRO_VALUE_TYPE_DOUBLE	Optional element. Specifies the offset along Z-axis. This element can range from negative to positive values. The valid range values for this element are from -MAX_DIM_VALUE to MAX_DIM_VALUE
PRO_E_TOOL_MTN_AXIS_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the axis selection. **Note** By default Z-axis of the Creo NC sequence is used to define the constraint plane.

Tool Motion — Chamfer Milling

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnChamferMill.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



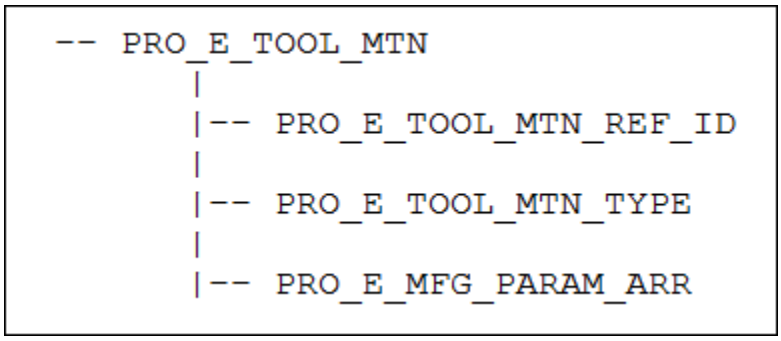
The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_CHAMFER_MILLING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Cutline Milling

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnCutlineMill.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



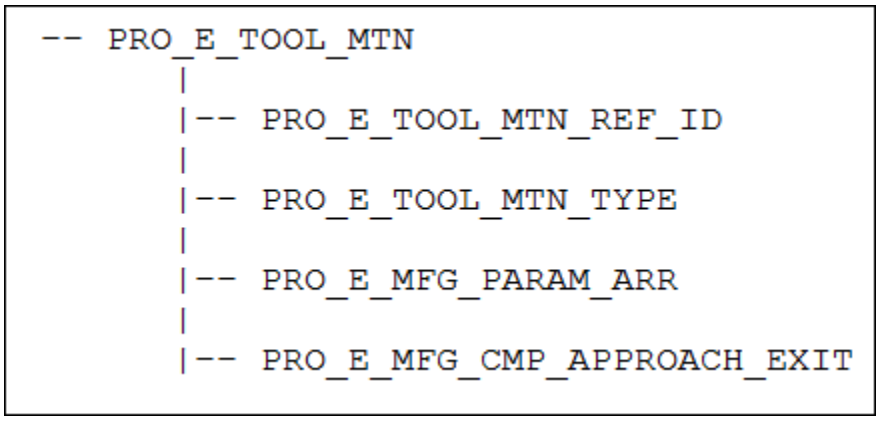
The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_CUTLINE_MILLING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Face Milling

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnFaceMill.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



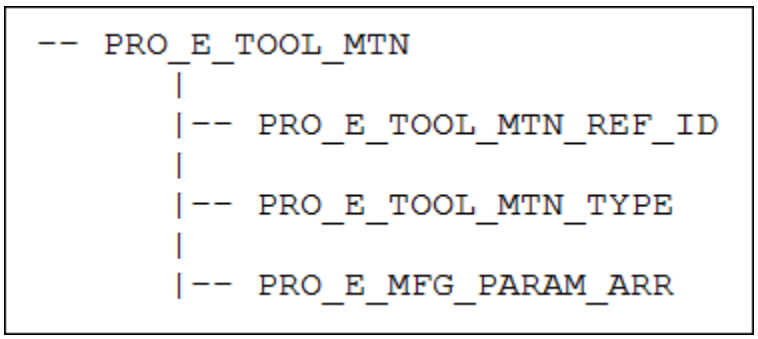
The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_FACE_MILLING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Optional element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree. **Note** This element is inherited from Creo NC Sequence if not specified.
PRO_E_MFG_CMP_APPROACH_EXIT	Compound	Optional element. Specifies the approach and exit compound definition. For more information, refer to the section Approach and Exit on page 1689.

Tool Motion — Groove Milling

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnGrooveMill.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



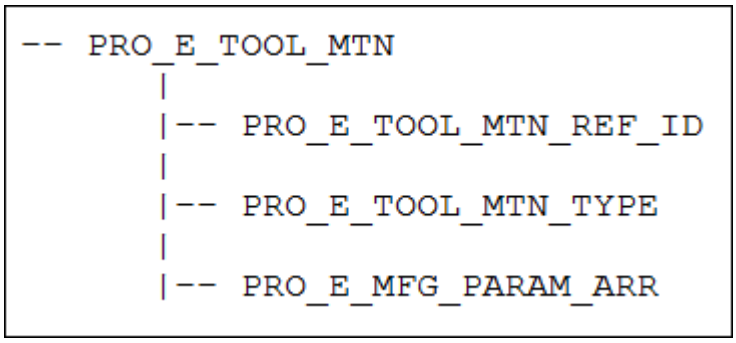
The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_GROOVE_MILLING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Round Milling

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnRoundMill.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



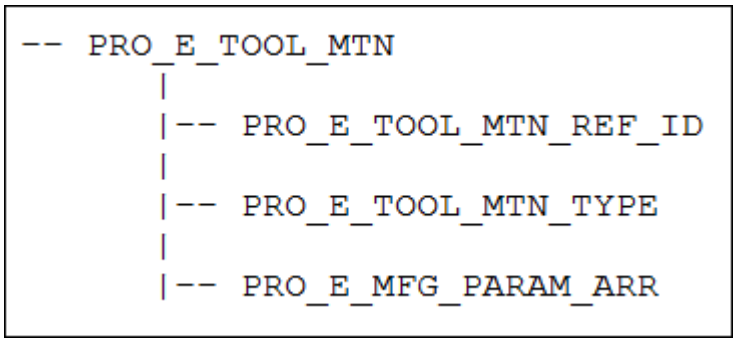
The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_ROUND_MILLING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Thread Milling

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnThreadMill.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element



The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the tool motion type . The valid value for this element is PRO_TM_TYPE_THREAD_MILLING. The value for this element is defined by ProTmType.
PRO_E_MFG_PARAM_ARR	Array	Mandatory element. Specifies an array of manufacturing parameters. See the Creo NC help for more information on manufacturing parameters. The element tree for the manufacturing parameter is defined in the header file ProMfgElemParam.h. For more information, refer to the section Manufacturing Parameters on page 1677 for more information on the element tree.

Tool Motion — Volume Mill Cut

The element PRO_E_TOOL_MTN is a member of PRO_E_TOOL_MTN_ARR array and is documented in the header file ProMfgElemToolMtnVolMillCut.h, and is shown in the following figure.

Element tree for PRO_E_TOOL_MTN element

```
-- PRO_E_TOOL_MTN
       |
       |-- PRO_E_TOOL_MTN_REF_ID
       |
       |-- PRO_E_TOOL_MTN_TYPE
       |
       |-- PRO_E_TOOL_MTN_PROFILE_TYPE
       |
       |-- PRO_E_MFG_PARAM_ARR
       |
       |-- PRO_E_MFG_CMP_APPROACH_EXIT
       |
       |-- PRO_E_MFG_START_HEIGHT
       |
       |-- PRO_E_MFG_HEIGHT
       |
       |-- PRO_E_VOL_MILL_REGION_ARR
              |
              |-- PRO_E_VOL_MILL_REGION
                     |
                     |-- PRO_E_VOL_MILL_REGION_IDX
                     |
                     |-- PRO_E_VOL_MILL_REGION_SKIP_OPT
                     |
                     |-- PRO_E_VOL_MILL_AP_REF
                     |
                     |-- PRO_E_VOL_MILL_EX_REF
```

The following table lists the contents of PRO_E_TOOL_MTN element.

Element ID	Data Type	Description
PRO_E_TOOL_MTN_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specify the tool motion type as PRO_TM_TYPE_VOLUME_MILLING. The value for this element is defined by the enumerated typeProTmType.
PRO_E_TOOL_MTN_PROFILE_TYPE	PRO_VALUE_TYPE_INT	Specifies the tool motion profile type. The enumerated type ProMillProfCutType defines the valid values for this element

Element ID	Data Type	Description
		which are as follows: • PRO_E_MILL_CUT_FULL_DEPTH— Specifies that the entire depth of the mill cut is machined. • PRO_E_MILL_CUT_FROM_TO— Specifies that the machining depth is limited by the specified start and the end references. • PRO_E_MILL_CUT_UPTO— Specifies that the machining depth is limited by the specified end reference. • PRO_E_MILL_CUT_ONE_SLICE— Specifies the machining of a single slice cut.
PRO_E_MFG_PARAM_ARR	Array	Optional element. Defines an array of manufacturing parameters. For more information, refer to the section Manufacturing Parameters on page 1677. This element is inherited from the Creo NC Sequence, if not specified.
PRO_E_MFG_CMP_APPROACH_EXIT	Compound	Specifies the approach and exit motions. This optional element is defined in the header file ProMfgElemApproachExit.h. For more information, refer to the section Approach and Exit on page 1689.
PRO_E_MFG_START_HEIGHT	PRO_VALUE_TYPE_SELECTION	Optional element. Select the horizontal surface from where machining will begin. **Note** This element is mandatory only if the element PRO_E_TOOL_MTN_PROFILE_TYPE is set to PRO_E_MILL_CUT_FROM_TO.
PRO_E_MFG_HEIGHT	PRO_VALUE_TYPE_SELECTION	Select a single surface where the

Element ID	Data Type	Description
		machining will begin. **Note** This element is mandatory only if the element PRO_E_TOOL_MTN_PROFILE_TYPE is set to PRO_E_MILL_CUT_FROM_TO, or PRO_E_MILL_CUT_UPTO, or PRO_E_MILL_CUT_ONE_SLICE.
PRO_E_VOL_MILL_REGION_ARR	Array	Optional element. Specifies an array of region specifications. Define this element to overwrite the default order of machining of different areas
PRO_E_VOL_MILL_REGION	Compound	Specifies a compound element that defines the volume milling region.
PRO_E_VOL_MILL_REGION_SKIP_OPT	PRO_VALUE_TYPE_INT	Specifies the skipping option for the region. The valid values for this element are: • PRO_B_TRUE—Specifies that the region will be skipped while machining. • PRO_B_FALSE—Specifies that the region will not be skipped while machining.
PRO_E_VOL_MILL_AP_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of a vertical datum axis, or a datum point. This element defines a location for the region to make an approach move.
PRO_E_VOL_MILL_EX_REF	PRO_VALUE_TYPE_SELECTION	Optional element. Specifies the selection of a vertical datum axis, or a datum point. This element defines a location for the region to make an exit move.

64

Production Applications: Process Planning

This chapter describes how to use the Creo Parametric TOOLKIT functions for assembly process operations. It assumes that you are familiar with the functionality of Manufacturing Process Planning for ASSEMBLIES.

Process Step Objects

Function Introduced:

- **ProProcstepInit()**

Process steps are represented by the object `ProProcstep`, which is an instance of `ProModelitem`. The object `ProProcstep` describes the contents and ownership of a assembly process step.

The declaration is as follows:

```
typedef struct pro_model_item
{
  ProType    type;
  int        id;
  ProMdl     owner;
} ProProcstep;
```

To create a new process step handle, use the function `ProProcstepInit()`.

Visiting Process Steps

Function Introduced:

- **ProProcstepVisit()**

The function `ProProcstepVisit()` enables you to visit all the process steps in the specified solid. For a detailed explanation of visiting functions, see the section Visit Functions on page 62 in the Fundamentals on page 22 chapter.

Process Step Access

Functions Introduced:

- **ProProcstepActiveGet()**
- **ProProcstepActiveSet()**
- **ProProcstepNumberGet()**

These functions access the process step objects.

The functions `ProProcstepActiveGet()` and `ProProcstepActiveSet()` enable you to get and set the current active process step.

The function `ProProcstepNumberGet()` retrieves the process step number for the specified solid and process step.

Creating Process Steps

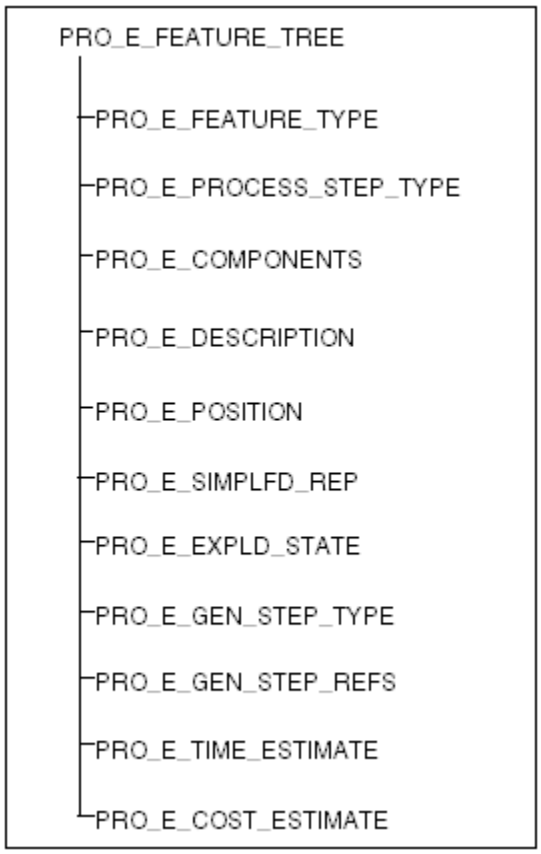
To create an assembly process step, use the function `ProFeatureCreate()`. As with any other type of feature, you use the element tree to create a process step feature.

This chapter describes the basic principles of programmatic process step creation. The chapter Element Trees: Principles of Feature Creation on page 764 is a necessary background for this topic; therefore, you should read that chapter first.

The element tree for a process step feature is documented in the header file `ProProcstep.h` and has a fairly simple structure, as shown in the following figure. You can also create a copy of the feature element tree of an existing process step feature by calling the function `ProFeatureElemtreeExtract()` with an input feature of type `ProProcstep`.

The following figure shows the element tree of a process step feature.

Element Tree of Process Step Feature



Feature Elements

The following table describes the tree elements in more detail.

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	PRO_FEAT_PROCESS_STEP
PRO_E_PROCESS_STEP_TYPE	PRO_VALUE_TYPE_INT	See Types of Process Step on page 1787
PRO_E_COMPONENTS	PRO_VALUE_TYPE_ SELECTION	Step components
PRO_E_DESCRIPTION	PRO_VALUE_TYPE_WSTRING	Step description
PRO_E_POSITION	PRO_VALUE_TYPE_ TRANSFORM	Position transformation
PRO_E_GEN_STEP_TYPE	PRO_VALUE_TYPE_WSTRING	General step type
PRO_E_GEN_STEP_REFS	PRO_VALUE_ TYPESELECTION_	General step references
PRO_E_SIMPLFD_REP	PRO_VALUE_TYPE_INT	Simplified representation identifier
PRO_E_EXPLODE_STATE	PRO_VALUE_TYPE_INT	Explode state identifier
PRO_E_TIME_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Time estimate (greater than 0.0)
PRO_E_COST_ESTIMATE	PRO_VALUE_TYPE_DOUBLE	Cost estimate (any value)

Types of Process Step

The types of process step are as follows:

- PRO_PROCSTEP_ASSEMBLE—Step to assemble components
- PRO_PROCSTEP_DISASSEMBLE—Step to disassemble components
- PRO_PROCSTEP_REASSEMBLE—Step to reassemble components
- PRO_PROCSTEP_REPOSITION—Step to reposition components
- PRO_PROCSTEP_GENERAL—General step (default or user-defined types)

Optional Elements

The following elements in the tree are optional for all types of process step:

- PRO_E_DESCRIPTION
- PRO_E_SIMPLFD_REP
- PRO_E_EXPLODE_STATE
- PRO_E_TIME_ESTIMATE
- PRO_E_COST_ESTIMATE

For a detailed explanation of explode states and related functions, see the section Exploded Assemblies on page 1141 in the Assembly: Basic Assembly Access on page 1130 chapter.

General Process Steps

The following two elements are used for general process steps (type PRO_PROCSTEP_GENERAL) only:

- PRO_E_GEN_STEP_TYPE—Default types, as well as user-defined types
- PRO_E_GEN_STEP_REFS—The following are valid reference types:
  - PRO_PART
  - PRO_FEATURE
  - PRO_SURFACE
  - PRO_EDGE
  - PRO_CURVE
  - PRO_AXIS
  - PRO_CSYS
  - PRO_POINT

Reposition Process Steps

The element PRO_E_POSITION is used for reposition process steps. It defines the transformation of the repositioned components.

65

Production Applications: NC Process Manager

Overview

The Process Manager functionality is based on the Process Table, which lists all the manufacturing process objects, such as workcells, operations, fixture setups, tooling, and NC sequences.

The Process Table lets you create new manufacturing objects and modify properties of existing ones. You can create new Holemaking and some milling steps directly in the Process Table. Steps of other types, that is, NC Sequences created outside the Process Manager, are also listed in the Process Table. You can manipulate the NC Sequences like any other steps, for example, reorder or duplicate them. However, customizations done to the toolpath outside the Process Table will be lost as a result of manipulation.

You can also define manufacturing templates based on existing Milling and Holemaking steps, and use these templates to create manufacturing steps in a different model.

You can add manufacturing templates as annotations to the reference part. This way, when you bring a reference part into a manufacturing model, the annotation can be automatically extracted into the manufacturing process to create the necessary steps, complete with tooling, parameters, and geometric references.

Accessing the Process Manager

You can access the Process Manager in Creo Parametric in the Manufacturing Mode via Creo Parametric TOOLKIT as follows:

- When the process manager dialog box is not active.
- When the process manager dialog box is active.

Accessing the Process Manager when the Dialog Box is not Active

You can access and modify the contents of the process manager without opening the Process Manager User Interface or while editing a manufacturing model in Creo Parametric.

Functions Introduced:

- **ProMfgProctableEnable()**
- **ProMfgProctableDisable()**

The function `ProMfgProctableEnable()` ensures that the process manager database is prepared for access or modification by Creo Parametric TOOLKIT functions. The application must call the function `ProMfgProctableEnable()` before using the process manager APIs discussed in the following sections.

The function `ProMfgProctableDisable()` enables Creo Parametric to perform cleanup of data that was loaded while calling the function `ProMfgProctableEnable()`.

Accessing the Process Manager User Interface

You can add new items to the Process Manager User Interface using a Creo Parametric TOOLKIT application. The Creo Parametric TOOLKIT application should register the new user interface items in Creo Parametric before you open the Process Manager dialog box. Once the dialog box is opened, Creo Parametric automatically adds the registered menu buttons and menus to the dialog box.

Functions Introduced:

- **ProMfgproctableMenuAdd()**
- **ProMfgproctablePushbuttonAdd()**
- **ProMfgproctableItemAccessFunction**
- **ProMfgproctableItemActionFunction**
- **ProMfgproctableSelecteditemsGet()**
- **ProMfgproctableSelecteditemsSet()**
- **ProMfgproctableDisplayUpdate()**

The function `ProMfgproctableMenuAdd()` adds a new menu or submenu to an existing menu in the process manager user interface.

The function `ProMfgproctablePushbuttonAdd()` adds a new button at the end of the existing menu in the Process Manager user interface. When adding a new menu button, you must supply two callback functions whose signature matches `ProMfgproctableItemAccessFunction` and `ProMfgproctableItemActionFunction` respectively.

The function whose signature matches `ProMfgproctableItemAccessFunction` sets the access state for the button in the process manager dialog box. The function whose signature is similar to `ProMfgproctableItemActionFunction` provides the implementation for the new menu button.

Use the functions `ProMfgproctableSelecteditemsGet()` and `ProMfgproctableSelecteditemsSet()` in the context of menu buttons added to the Process Manager. These functions provide access to the steps, operations and workcells that are currently selected by the user in the Process Manager dialog box. This information can guide the application as to which items it should modify. Select the appropriate type of table for the selected item to be able to determine the subtype of the process item.

These functions can be used out of context of the menu addition APIs as well.

The function ProMfgproctableDisplayUpdate() refreshes the Process Manager dialog box. It allows the application to update the contents of the process table after some changes have been made during a custom menu button action.

Manufacturing Process Items

This section gives you more information on the handles that are provided to enable access to process item data.

Steps, Operations and Workcells

An NC sequence is an assembly feature that represents a single tool path. NC sequences listed in the Process Table are called steps.

An operation is a series of NC sequences performed in a particular workcell and using a particular coordinate system for output of Cutter Location data.

A workcell is a machine tool definition and is stored in the assembly.

The object ProMfgprocItem will be used for all functions that operate on steps, operations, and workcells. This item is a derivative of ProModelitem and is defined as

```
typedef struct pro_model_item
{
ProType type;
int id;
ProMdl owner;
}ProMfgprocItem;
```

For all three item types, the ProType field should be PRO_NC_PROCESS_ITEM.

Step Models

The Step Model object represents the default properties of steps in the process manager and is a derivative of the ProModelitem and is defined as typedef struct pro_model_item.

```
{
ProType type;
int id;
ProMdl owner;
}ProMfgprocItem;
```

Step objects have a step id, the Mfg owner, and the ProType field as PRO_NC_PROCESS_MODEL.

Determining the Subtype of a Process Item

Function Introduced:

- **ProMfgprocitemSubtypeGet()**

The function `ProMfgprocitemSubtypeGet()` returns the subtype of a manufacturing process item. It could be one of the following:

- `PRO_MFGPROCITEM_STEP`—Specifies if the subtype is a step.
- `PRO_MFGPROCITEM_OPERATION`—Specifies if the subtype is an operation.
- `PRO_MFGPROCITEM_WORKCELL`—Specifies if the subtype is a workcell.

Accessing Details of Process Items

Although a `ProMfgprocItem` is not a feature, it shares many properties with Creo Parametric TOOLKIT features. In fact, process items generally map one-to-one with a specified manufacturing feature. Because of this, many of the process item's properties can be accessed via the corresponding manufacturing feature element tree. Also, new items may be created by populating the element tree for the new step, operation or workcell.

Visiting Steps, Operations and Workcells

Functions Introduced:

- **ProMfgProctableVisit()**
- **ProMfgprocitemVisitAction()**
- **ProMfgprocitemFilterAction()**

The function `ProMfgProctableVisit()` enables you to visit the steps, workcells, and operations in a process table. As with other Creo Parametric TOOLKIT visit functions, specify the visit action and visit filter functions. For more information on Visit Functions, refer to the chapter Fundamentals on page 22.

Creating Steps, Operations, and Workcells

Functions Introduced:

- **ProMfgprocitemCreate()**
- **ProMfgprocitemFromTemplateCreate()**

Use the function `ProMfgprocitemCreate()` to create a new step, operation, or workcell in the process table. The input arguments of this function are:

- *model*—Specifies the manufacturing model in which to create the new item.
- *table_type*—Specifies the type of table where the item will reside. The valid values are:
  - `PRO_MFGPROCTABLE_PROCESS`—Specifies the process table.
  - `PRO_MFGPROCTABLE_WORKCELLS`—Specifies the workcell table.
- *elem_tree*—Specifies the manufacturing feature element tree used to enumerate the properties of the new manufacturing item. To create a step you must fill one of the following element trees:
  - Milling and Drilling—The element tree documented in header file `ProNcseq.h`.
  - Fixture—The element tree documented in header file `ProFixture.h`.
  - Assembly setup step—The element tree documented in header file `ProFixture.h`.

  To create an Operation you must fill out the element tree documented in header file `ProMfgoper.h`.

  To create a Workcell you must fill the element tree documented in header file `ProWcell.h`.
- predecessor—Specifies the step or operation immediately before the new manufacturing item in the process table. When creating workcells, this can be `NULL`.

The function `ProMfgprocitemCreate()` returns the new manufacturing item.

Use the function `ProMfgprocitemFromTemplateCreate()` to create a new manufacturing step or workcell from a predefined template. A template is an XML file containing information necessary to create the manufacturing step or workcell, such as:

- Resources, that is, workcells, tools, and fixtures
- Ordered operations
- Ordered steps and their reference information such as manufacturing templates and manufacturing criteria.
- Retract plane for a step or an operation, as a distance calculated from the step coordinate system.

Accessing the Properties of Manufacturing Process Items

The functions described below enable you to access the properties of process items.

Functions Introduced:

- **ProMfgprocitemFeatureGet()**
- **ProMfgprocitemElemtreeGet()**
- **ProMfgprocitemElemtreeFree()**
- **ProMfgprocitemAnnotationGet()**
- **ProMfgprocitemReorderlimitsGet()**
- **ProMfgprocitemNextitemGet()**
- **ProMfgprocitemPreviousitemGet()**
- **ProMfgprocitemHolesetdepthGet()**
- **ProMfgprocitemHolesetdepthtypeGet()**
- **ProMfgprocitemHolesetdepthSet()**
- **ProMfgprocitemDefaultfixturesetupstepGet()**

The function `ProMfgprocitemFeatureGet()` returns the feature referenced by the process item. If the item is related to an actual manufacturing feature, some of the access functions that follow will not be supported. The properties of the item can be obtained by reading the feature's properties and element tree.

The function `ProMfgprocitemElemtreeGet()` obtains the element tree for a manufacturing item. This tree should be freed using `ProMfgprocitemElemtreeFree()`.

The function `ProMfgprocitemAnnotationGet()` returns the annotation element that created the process item.

The function `ProMfgprocitemReorderlimitsGet()` identifies the limits where the specified process item may be moved in the process table. The top limit represents the first process item after which you can insert the step. The lower limit, if any, represents the first process item that must come after the step in the process sequence.

The limits of reorder are based on step dependencies and priority and prerequisite rules. The input argument *options* specifies the checks to determine the position of reordering the process item elements. The valid values are:

- `PRO_MFGPROCREORDER_CHECK_PARENT_CHILD`
- `PRO_MFGPROCREORDER_CHECK_PREREQUISITES`
- `PRO_MFGPROCREORDER_CHECK_PRIORITIES`

The function `ProMfgprocitemNextitemGet()` returns the process item following the specified item in the process table.

The function `ProMfgprocitemPreviousitemGet()` returns the process item preceding the specified process item in the process table.

The function ProMfgprocitemHolesetdepthtypeGet() returns the type of holeset and holeset end reference contained in the manufacturing process step.

The output argument *end_type* can have the following values:

- PRO_MFGSTEP_HOLESETEND_REFERENCE—Specifies the end reference surface for the hole depth.
- PRO_MFGSTEP_HOLESETEND_ALONG_AXIS—Specifies the end reference by entering a depth value along the hole axis.

The function ProMfgprocitemHolesetdepthGet() returns the process item holeset depth for a holeset of type BLIND and end reference type of ALONG AXIS.

The function ProMfgprocitemHolesetdepthSet() sets the holeset depth for a step. This function supports steps with only one holeset. The holeset type must be BLIND and the end reference type will be changed to ALONG AXIS after using this function.

The function ProMfgprocitemDefaultfixturesetupstepGet() returns the fixture setup defined for the specified operation.

Modifying Process Items

Functions Introduced:

- **ProMfgprocitemRedefine()**
- **ProMfgprocitemDelete()**
- **ProMfgprocitemReorder()**

Use the function ProMfgprocitemRedefine() to redefine the manufacturing item created using an element tree.

The function ProMfgprocitemDelete() deletes the specified step, operation, or workcell from the manufacturing table.

The function ProMfgprocitemReorder() reorders a step or operation within the process table.

Parameters

Several categories of parameters are accessible for items in the process manager table:

- Manufacturing parameters
- AE parameters
- Step parameters
- Template parameters

- Special parameters
- Global Parameters

Manufacturing Parameters

You can access the manufacturing parameters by reading or writing the manufacturing feature or step element tree.

Annotation Element Parameters

You can access the annotation element parameters by getting the parent Annotation Element via `ProMfgprocitemAnnotationGet()` and reading parameters from that object. Refer to the chapter Parameters for details on how to read parameters from items such as annotation elements.

Step Parameters and Relations

Step Parameters are parameters assigned to the step object. You can define parameters and relations to a particular step in the process table.

You can access these parameters by reading them directly from the step object using the standard parameter functions.

Refer to the chapter Parameters for details on how to read parameters from items.

Template Parameters

Template parameters are inherited from the step's process template.

Function Introduced:

- **ProMfgprocitemTemplateparamGet()**

The function `ProMfgprocitemTemplateparamGet()` obtains a template parameter value for the process item.

Special Parameters

Other special parameters are properties of steps and can include the extract status, machining time, the template name, and so on. Some of these properties may be also accessible via the step element tree.

Functions Introduced:

- **ProMfgprocitemPropertyGet()**
- **ProMfgprocitemPropertySet()**

The function ProMfgprocitemPropertyGet() returns the value of the special property of the process manager step or operation. The special property can be as follows:

- PRO_MFGPROP_EXTRACT_STATUS—Specifies the extraction priority of the step.
- PRO_MFGPROP_ACTUAL_MACHINING_TIME—Specifies the machining time of a step.
- PRO_MFGPROP_ACTUAL_MACHINING_LENGTH— Specifies the machining distance of a step.
- PRO_MFGPROP_TEMPLATE_NAME—Specifies the name of the manufacturing template used.
- PRO_MFGPROP_GROUP_LEVEL_1—Specifies a merge group created by merging several Holemaking steps together, to optimize the tool path.
- PRO_MFGPROP_GROUP_LEVEL_2—Specifies a merge group created by merging PRO_MFGPROP_GROUP_LEVEL_1 type merge groups together.

ProMfgprocitemPropertyGet() returns string values for PRO_MFGPROP_GROUP_LEVEL_1 and PRO_MFGPROP_GROUP_LEVEL_2 depending on the location of the process manager step in the merged group. The following table lists all the possible values taken by the special properties.

Step Location	Value returned for	Value returned for
If the step is a merge leader, but not a merge member.	"*leader"	Empty string
If the step is both a merge leader and merge member.	Name of the parent merge leader	"*leader"
If the step is merge member, but not a merge leader.		
Case 1: It is a two level merge. For example, the parent itself is a merge member.	Name of the top merge leader	Name of the parent merge leader
Case 2: It is a one level merge	Name of the parent merge leader	Empty string
If the step is neither merge leader nor a merge member.	Empty string	Empty string

The function ProMfgprocitemPropertySet() sets the value of the special property of the process manager step or operation.

Global Parameters and Relations

Function Introduced:

- **ProMfgProctablemodelGet()**

The function ProMfgProctablemodelGet() obtains the process model that provides access to the manufacturing global parameters.

You can use this function to specify parameters and relations to all or selected steps in the process table. The `modelitem` handle returned by this function can be used to access the Global parameters of the process table.

Manufacturing Features

Functions Introduced:

- **ProMfgproctableFeaturesCreate()**
- **ProMfgproctableFeaturesDelete()**

The function `ProMfgproctableFeaturesCreate()` creates features for items in the process manufacturing table.

The function `ProMfgproctableFeaturesDelete()` deletes all existing manufacturing features from the Creo Parametric model while keeping everything in the process manager unchanged.

Import and Export of Process Table Contents

Functions Introduced:

- **ProMfgProctableWrite()**
- **ProMfgProctableSynchronize()**

The function `ProMfgProctableWrite()` exports the contents of the manufacturing process table to CSV format. The input arguments of the function are:

- *mfg*—Specifies the manufacturing model.
- *table_type*—Specifies the type of table to be generated as output. The valid values are:
  - `PRO_MFGPROCTABLE_PROCESS`
  - `PRO_MFGPROCTABLE_WORKCELLS`
- *view_name*—Specifies the view name in the setup file; pass `NULL` to use the current view.
- *output_file*—Specifies the full path and name of the output file.

The function `ProMfgProctableSynchronize()` synchronizes the manufacturing table with the input CSV file.

Notification

Function Introduced:

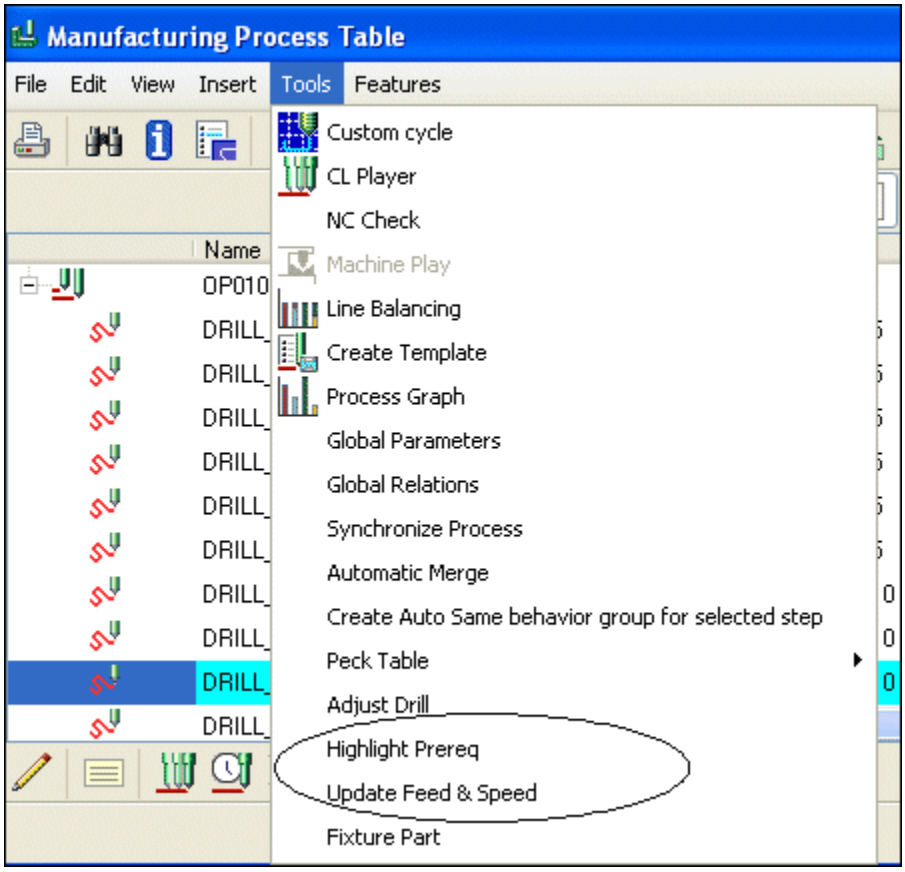
- **ProMfgproctableExtractionPostAction**

The notification `ProMfgproctableExtractionPostAction` identifies when the user has extracted the process table in order to form manufacturing features. It provides the path to the extraction log file, which can be examined by the application for further information.

Example 1: To Add Menu Button to The Manufacturing Process Table

This example enables you to add the Highlight Prereq and the Update Speed and Feed options respectively, to the **Tools** menu in the Manufacturing Process Table.

```
/*====================================================================*\
  FUNCTION : user_initialize()
  PURPOSE  :
\*====================================================================*/
int user_initialize(
                    int argc,
                    char *argv[],
                    char *version,
                    char *build,
                    wchar_t errbuf[80])
{
  uiCmdCmdId    cmd_id;
  PTUtilInitCounters();

  status = ProMfgproctablePushbuttonAdd ("Tools","Highlight Prereq",
                                          L"Highlight Prereq",
                                         L"button_helptext",NULL,
                                         PTMfgProctableItemActionPreReq,
                "Callback on the new submenu added in
  existing menu");
  PT_TEST_LOG_SUCC("ProMfgproctablePushbuttonAdd");


  status = ProMfgproctablePushbuttonAdd ("Tools","Update Feed & Speed",
                                          L"Update Feed & Speed",
                                         L"button_helptext",NULL,
                                         PTMfgProctableItemFeedSpeed,
              "Callback on the new submenu added in
  existing menu");
  PT_TEST_LOG_SUCC("ProMfgproctablePushbuttonAdd");

return (PRO_TK_NO_ERROR);
}
```



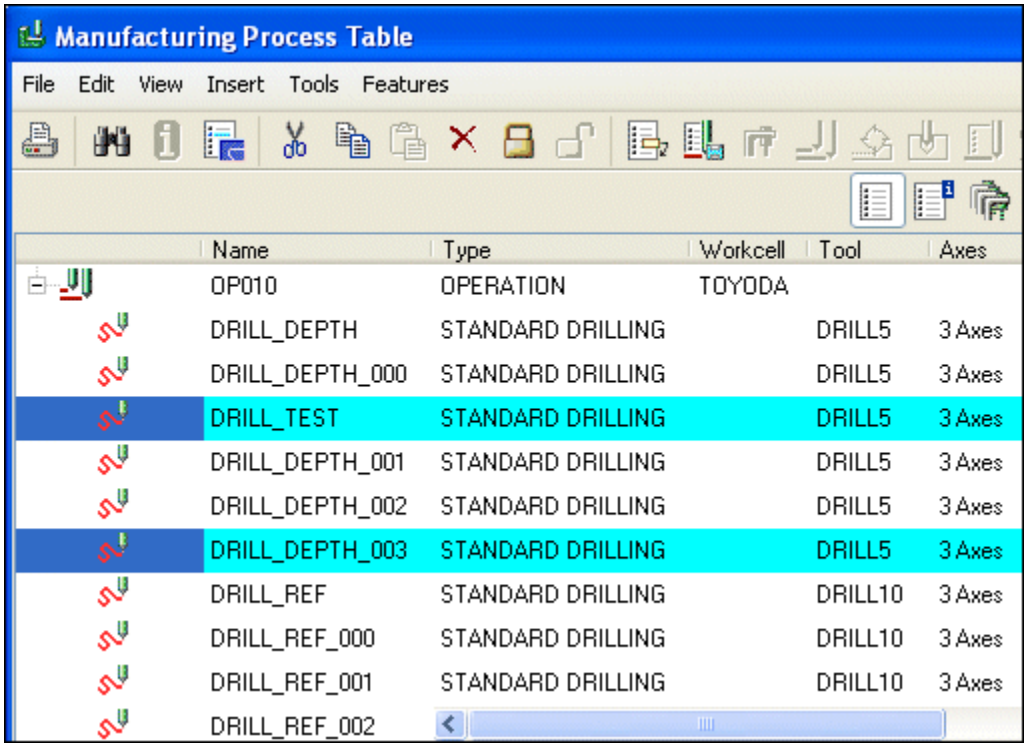
The image above displays the Highlight Prereq and Update Feed & Speed menu buttons added to the **Tools** menu in the Manufacturing Process Table. When you click the Highlight Prereq option the function `PTMfgProctableItemActionPreReq` is invoked. The following example deals with this function in detail.

Example 2: To Highlight the Pre-requisites for the Selected Step in The Manufacturing Process Table

This example enables you to highlight the pre-requisites for the selected step in the Manufacturing Process Table.

```
/*=====================================================================*\
    FUNCTION :  PTMfgProctableItemActionPreReq
    PURPOSE  :  Execute Highlighting of Pre-Requisite steps
\*=====================================================================*/
ProError PTMfgProctableItemActionPreReq (ProMfg mfg, char* button_name,
            ProAppData appdata)
{

  ProElement elem_tree;
```

```
  status = PTMfgProcitemElemtreeGet(&elem_tree);
  PT_TEST_LOG_SUCC("PTMfgProcitemElemtreeGet");

  if ( status == PRO_TK_NO_ERROR)
    {
      status = ProElemtreeWrite (elem_tree, PRO_ELEMTREE_XML,
           L"selecteditem_prereq.inf");
      PT_TEST_LOG_SUCC (" ProElemtreeWrite ");

      if ( status == PRO_TK_NO_ERROR )
      {
        status = ProElemtreeElementVisit( elem_tree, ( ProElempath )NULL ,
                       ( ProElemtreeVisitFilter) NULL,
                       ( ProElemtreeVisitAction) PTMfgElemtreePreReqAction,
                                 NULL );
        PT_TEST_LOG_SUCC (" ProElemtreeElementVisit ");
      }

    }

  return PRO_TK_NO_ERROR;
}
```

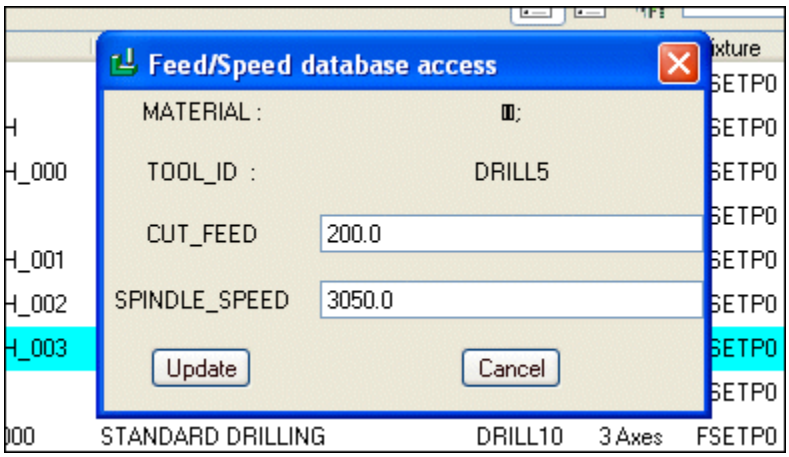


Example 3: To Update Feed and Spindle Speed

When you click the **Update Speed and Feed** options in the **Tools** menu in the Manufacturing Process Table, the function `PTMfgProctableItemFeedSpeed` is invoked. This sample code enables you to change the cut feed and the spindle speed for the specified tool.

```
/*====================================================================*\
    FUNCTION :  PTMfgProctableItemFeedSpeed()
    PURPOSE  :  Execute Feed/Speed
\*====================================================================*/
ProError PTMfgProctableItemFeedSpeed (ProMfg mfg, char* button_name
, ProAppData appdata)
{
  ProAssembly r_solid_obj;
  ProMdl mfg_mdl_under_test;
  ProElempath epath;
  ProElement params_element;
  ProElempathItem  p_items[3];
  ProElemId elem_id;
  ProValueDataType value_type = -1;
  ProValue     value = (ProValue)NULL;
  ProValueData value_data;
  status =ProMdlCurrentGet(&mfg_mdl_under_test);
  PT_TEST_LOG_SUCC("ProMdlCurrentGet");

  status = PTMfgProcitemElemtreeGet(&complete_tree);
  PT_TEST_LOG_SUCC("PTMfgProcitemElemtreeGet");

  if (status  == PRO_TK_NO_ERROR)
    PTMfgProcitemToolNameGet(complete_tree);

  if  (status  == PRO_TK_NO_ERROR)
    {
      status = ProElempathAlloc(&epath);
      PT_TEST_LOG_SUCC ("ProElempathAlloc ()");

      p_items[0].type = PRO_ELEM_PATH_ITEM_TYPE_ID;
      p_items[0].path_item.elem_id = PRO_E_MFG_PARAMS ;

      status = ProElempathDataSet(epath, p_items, 1);
      PT_TEST_LOG_SUCC ("ProElempathDataSet()");
      status= ProElemtreeElementGet(complete_tree, epath,
&params_element);
      PT_TEST_LOG_SUCC ("ProElemtreeElementGet()");

      if ( status == PRO_TK_NO_ERROR )
      {
        status = ProElemtreeElementVisit( params_element,NULL,
                                  ( ProElemtreeVisitFilter) NULL,
                                  ( ProElemtreeVisitAction)
                                  PTMfgElemtreeWalkthroughAction,
```

```
                                    NULL );
        PT_TEST_LOG_SUCC (" ProElemtreeElementVisit ");
      }
      ProElempathFree(&epath);    }
    status = ProMfgAssemGet(mfg_mdl_under_test,&r_solid_obj);
  PT_TEST_LOG_SUCC ("ProMfgAssemGet");
    status = ProSolidFeatVisit ((ProSolid)r_solid_obj,
                                (ProFeatureVisitAction)PTMfgMaterialGet,
                                NULL,
                                NULL);
                                PT_TEST_LOG_SUCC ("ProSolidFeatureVisit()");

  status  = PTMfgDialogCreate(params_element);
    PTUtilInitCounters();
    return PRO_TK_NO_ERROR;
}
```



The section below describes the action functions for the each of the preceding sample code.

```
/************************ACTION FUNCTIONS************************/
/*-----------------------------------------------------------*\
    FUNCTION :  PTMfgElemtreeWalkthroughAction()
    PURPOSE  :  Action function for PTUtilElemtreeWalkthroug
\*-----------------------------------------------------------*/
ProError PTMfgElemtreeWalkthroughAction ( ProElement params_element,
                                ProElement  element, ProElempath elem_path,
                                ProAppData app_data )
{    ProElempathItem  p_items[3];
  ProValueDataType value_type = -1;
  ProElemId  elem_id,p_elem_id;  char *  actual_value;
  ProValue     value = (ProValue)NULL;
  ProValueData value_data;  Data_Input *my_data;
  ProElempath      epath;  my_data = (Data_Input*) app_data;
  status = ProElementIdGet( element, &elem_id);
  PT_TEST_LOG_SUCC("ProElementIdGet");
    if ( elem_id == PRO_E_MFG_PARAM_NAME )
```

```
{       status = ProElementValuetypeGet ( element, &value_type );
  PT_TEST_LOG(" ProElementValuetypeGet ", status,
           status != PRO_TK_NO_ERROR && status != PRO_TK_INVALID_TYPE &&
           status != PRO_TK_EMPTY  );
        if(value_type ==PRO_VALUE_TYPE_STRING)
  {       status = ProElementStringGet( element,( ProElementStringOptions)
           NULL,
           &actual_value);
      PT_TEST_LOG_SUCC (" ProElementStringGet ");
             if (  (strcmp ( "CUT_FEED",actual_value)      == 0)
        ||( strcmp ( "SPINDLE_SPEED",actual_value) == 0))
        {
        status = ProElempathAlloc(&epath);
          PT_TEST_LOG_SUCC ("ProElempathAlloc ()");
                    p_items[0].type = PRO_ELEM_PATH_ITEM_TYPE_INDEX;
        p_items[0].path_item.elem_index = count;
        p_items[1].type = PRO_ELEM_PATH_ITEM_TYPE_ID;
        p_items[1].path_item.elem_id = PRO_E_MFG_PARAMVAL;
                    status = ProElempathDataSet(epath, p_items, 2);
       PT_TEST_LOG_SUCC ("ProElempathDataSet()");
        status= ProElemtreeElementGet(params_element, epath, &p_element);
       PT_TEST_LOG_SUCC ("ProElemtreeElementGet()");
                    status = ProElementIdGet( p_element, &p_elem_id);
        PT_TEST_LOG_SUCC (" ProElementIdGet ");
                    if (p_elem_id ==PRO_E_MFG_PARAMVAL)
        {
          status = ProElementValueGet ( p_element, &value);
          PT_TEST_LOG_SUCC("ProElementValueGet");
                        status = ProValueDataGet (value, &value_data);
          PT_TEST_LOG("ProValueDataGet()", status,
                      (status != PRO_TK_NO_ERROR));
                           if (strcmp ( "CUT_FEED",actual_value) == 0)
            {                 cut_feed = value_data.v.d;
                               if (FLAG == 1)
              {                   value_data.v.d = my_data-cut_feed;
                status = ProValueDataSet (value, &value_data);
                PT_TEST_LOG("ProValueDataSet()", status,
                (status != PRO_TK_NO_ERROR));
                 status = ProElementValueSet ( p_element, value);
                PT_TEST_LOG_SUCC("ProElementValueGet");
                                  }
            }
              if (strcmp ( "SPINDLE_SPEED",actual_value) == 0)
            {                 spindle_speed = value_data.v.d;
                               if (FLAG == 1)
              {                 value_data.v.d = my_data-spindle_speed;
                status = ProValueDataSet (value, &value_data);
                PT_TEST_LOG("ProValueDataSet()", status,
                        (status != PRO_TK_NO_ERROR));
                                    status = ProElementValueSet
                         (p_element, value);
```

```
                    PT_TEST_LOG_SUCC("ProElementValueGet");
                                        }
                }               count++;
            }
            ProElempathFree(&epath);
                     }
      }
            status =  ProMfgproctableDisplayUpdate ();
      PT_TEST_LOG_SUCC ("ProMfgproctableDisplayUpdate");
          }
    return PRO_TK_NO_ERROR;
}
/*--------------------------------------------------------*\
  FUNCTION : PTMfgUsrUpdateAction
  PURPOSE  : Action function for the File Selection
\*--------------------------------------------------------*/
void PTMfgUsrUpdateAction(char *dialog, char *component, ProAppData data)
{  ProElement params_element;
  ProCharName str;  ProCharName str2;
  double cut_feed_value;  double spindle_speed_value;
  Data_Input data1;  wchar_t* wstr;
  wchar_t* wstr2;  ProErrorlist    p_errors;
  params_element = (ProElement) data;
    ProUIInputpanelValueGet(DIALOG_NAME,"cut_feed",&wstr);
  ProWstringToString(str, wstr);
  cut_feed_value= atof(str);
  ProUIInputpanelValueGet(DIALOG_NAME,"spindle_speed",&wstr2);
  ProWstringToString(str2, wstr2);
  spindle_speed_value= atof(str2);
  data1.cut_feed = cut_feed_value;
  data1.spindle_speed = spindle_speed_value;
   count =0;  FLAG =1;
   status = ProElemtreeWrite (params_element, PRO_ELEMTREE_XML,
 L"before_redef.inf");
  PT_TEST_LOG_SUCC (" ProElemtreeWrite ");
      status = ProElemtreeElementVisit( params_element, ( ProElempath )NULL ,
                          ( ProElemtreeVisitFilter) NULL,
                          ( ProElemtreeVisitAction)
PTMfgElemtreeWalkthroughAction,
                          (ProAppData)&data1 );
  PT_TEST_LOG_SUCC (" ProElemtreeElementVisit ");
      status = ProMfgprocitemRedefine (sel_items,complete_tree,&p_errors);
  PT_TEST_LOG_SUCC ("ProMfgprocitemRedefine");
    status =  ProMfgproctableDisplayUpdate ();
  PT_TEST_LOG_SUCC ("ProMfgproctableDisplayUpdate");
    PTUtilInitCounters();
    status = ProUIDialogExit(DIALOG_NAME, 0);
  PT_TEST_LOG_SUCC("ProUIDialogExit");
}
/*--------------------------------------------------------*\
  FUNCTION : PTMfgUsrCancelAction
```

```
  PURPOSE  : Action function for the Directory Selection
\*--------------------------------------------------------*/
void PTMfgUsrCancelAction(char *dialog, char *component, ProAppData data)
{  ProError status;
    PTUtilInitCounters();
    status = ProUIDialogExit(DIALOG_NAME, 0);
  PT_TEST_LOG_SUCC("ProUIDialogExit");
  }
/*--------------------------------------------------------*\
    FUNCTION :  PTMfgElemtreePreReqAction()
    PURPOSE  :  Get the material
\*--------------------------------------------------------*/
ProError PTMfgMaterialGet (ProFeature* feat, ProAppData data)
{  ProAsmcompType f_type;
  ProMaterial p_material;  ProFeattype  feat_type;
  ProSolid part;  ProMdl p_mdl_handle;
  ProCharName c_name;  ProMdl mfg_mdl_under_test;
    status =ProMdlCurrentGet(&mfg_mdl_under_test);
  PT_TEST_LOG_SUCC("ProMdlCurrentGet");
    status = ProFeatureTypeGet(feat,&feat_type);
  PT_TEST_LOG_SUCC("ProFeatureTypeGet");
    if(feat_type==PRO_FEAT_COMPONENT)
    {    status = ProAsmcompTypeGet ((ProAsmcomp*)feat,feat-owner, &f_type);
      PT_TEST_LOG_SUCC("ProAsmcompTypeGet");
            if ( f_type ==  PRO_ASM_COMP_TYPE_REF_MODEL)
      {        status = ProAsmcompMdlGet ((ProAsmcomp*)feat,&p_mdl_handle);
        PT_TEST_LOG_SUCC("ProAsmcompMdlGet");
                status = ProMaterialCurrentGet
                ((ProSolid)p_mdl_handle,&p_material);
        PT_TEST_LOG_SUCC("ProMaterialCurrentGet");
                ProWstringToString (c_name, p_material.matl_name);
                ProWstringCopy(p_material.matl_name,material_name,
                 PRO_VALUE_UNUSED);
              }
          }
  return PRO_TK_NO_ERROR;}
/*--------------------------------------------------------*\
    FUNCTION :  PTMfgElemtreePreReqAction()
    PURPOSE  :  Action function for PTUtilElemtreeWalkthroug
\*--------------------------------------------------------*/
ProError PTMfgElemtreePreReqAction ( ProElement elem_tree,
                             ProElement  element, ProElempath elem_path,
                             ProAppData app_data )
{  ProElemId  elem_id;
  ProValueData value_data;  int           ii, counter = 0;
  ProValue      *values = (ProValue *)NULL;
  ProMfgprocItem *item_array;  ProMfgprocItem item;
  ProAssembly r_solid_obj;  ProElement elem_tree_p;
    ProMdl mfg_mdl_under_test;
    status =ProMdlCurrentGet(&mfg_mdl_under_test);
  PT_TEST_LOG_SUCC("ProMdlCurrentGet");
```

```
    status = ProMfgAssemGet(mfg_mdl_under_test,&r_solid_obj);
  PT_TEST_LOG_SUCC ("ProMfgAssemGet");
    status = ProElementIdGet( element, &elem_id);
  PT_TEST_LOG_SUCC("ProElementIdGet");
      if (elem_id == PRO_E_NCSEQ_PREREQUISITE_ARR)
    {      status = ProArrayAlloc( 0, sizeof(ProValue), 1,
(ProArray *)&values );
      PT_TEST_LOG_SUCC("ProArrayAlloc");
            status   = ProArrayAlloc( 0, sizeof(ProMfgprocItem), 1, (ProArray *)
&item_array );
      PT_TEST_LOG_SUCC("ProArrayAlloc(");
            status = ProElementValuesGet(element, &values );
      PT_TEST_LOG_SUCC("ProArrayAlloc");
            if( status  == PRO_TK_NO_ERROR )
        status  = ProArraySizeGet( (ProArray)values, &counter );
            for( ii = 0; ii < counter; ii++ )
        {
          if( ( ProValueDataGet( values[ii], &value_data ) == PRO_TK_NO_ERROR )
          && ( value_data.type == PRO_VALUE_TYPE_INT ) )
          {            item.type = PRO_NC_STEP_OBJECT;
          item.id  = value_data.v.i;
          item.owner  = r_solid_obj;          }
             status = ProArrayObjectAdd((ProArray*)&item_array,-1,1,&item);
          PT_TEST_LOG_SUCC("ProArrayObjectAdd");
        }
  status = ProMfgproctableSelecteditemsSet (PRO_MFGPROCTABLE_PROCESS,item_array);
      PT_TEST_LOG_SUCC ("ProMfgproctableSelecteditemsSet");
            status =  ProMfgproctableDisplayUpdate ();
      PT_TEST_LOG_SUCC ("ProMfgproctableDisplayUpdate");
            ProArrayFree ( (ProArray *)&values );
      ProArrayFree ( (ProArray *)&item_array );
    }    return PRO_TK_NO_ERROR;
}/*--------------------------------------------------------*\
    FUNCTION : PTMfgDialogCreate
    PURPOSE  : Creates the dialog
\*--------------------------------------------------------*/
ProError PTMfgDialogCreate(ProElement params_element)
{    int choice;
  ProCharName str;  ProName wstr;
  status = ProUIDialogCreate(DIALOG_NAME,DIALOG_NAME);
  PT_TEST_LOG_SUCC("ProUIPTMfgDialogCreate");
    status = ProUIDialogWidthSet(DIALOG_NAME,300);
  PT_TEST_LOG_SUCC("ProUIPTMfgDialogCreate");
    status = ProUIDialogHeightSet(DIALOG_NAME,160);
  PT_TEST_LOG_SUCC("ProUIPTMfgDialogCreate");
    status = ProUIDialogTitleSet (DIALOG_NAME,L"Feed/Speed database access");
  PT_TEST_LOG_SUCC ("ProUIDialogTitleSet()");
    status = ProUILabelTextSet (DIALOG_NAME, "tool_id_val", tool_id);
  PT_TEST_LOG_SUCC ("ProUILabelTextSet()");
    status = ProUILabelTextSet (DIALOG_NAME, "material_val", material_name);
  PT_TEST_LOG_SUCC ("ProUILabelTextSet()");
```

```
  sprintf(str,"%f",cut_feed);  ProStringToWstring(wstr, str);
  ProUIInputpanelValueSet(DIALOG_NAME,"cut_feed",wstr);
  PT_TEST_LOG_SUCC("ProUIInputpanelValueSet");
  sprintf(str,"%f",spindle_speed);
  ProStringToWstring(wstr, str);
  ProUIInputpanelValueSet(DIALOG_NAME,"spindle_speed",wstr);
  PT_TEST_LOG_SUCC("ProUIInputpanelValueSet");
  status = ProUIPushbuttonActivateActionSet
  (DIALOG_NAME, UPDATE, PTMfgUsrUpdateAction, params_element);
  PT_TEST_LOG_SUCC("ProUIPushbuttonActivateActionSet");
  status = ProUIPushbuttonActivateActionSet
  (DIALOG_NAME, CANCEL, PTMfgUsrCancelAction, NULL);
  PT_TEST_LOG_SUCC("ProUIPushbuttonActivateActionSet");
  status = ProUIDialogActivate(DIALOG_NAME, &choice);
  PT_TEST_LOG_SUCC("ProUIDialogActivate");
  status = ProUIDialogDestroy(DIALOG_NAME);
  PT_TEST_LOG_SUCC("ProUIDialogDestroy");
}
/*--------------------------------------------------------*\
    FUNCTION : PTMfgProcitemElemtreeGet
    PURPOSE  : Gets the elem tree of selected item
\*--------------------------------------------------------*/
 ProError  PTMfgProcitemElemtreeGet(ProElement *elem_tree)
 {   status = ProMfgproctableSelecteditemsGet
     (PRO_MFGPROCTABLE_PROCESS,&sel_items);
   PT_TEST_LOG_SUCC ("ProMfgproctableSelecteditemsGet");
         /*TBD : Currently one one ( fitst item is checked here...
     To be updated with multiple selections of items in ui ..*/
      status = ProMfgprocitemElemtreeGet (&sel_items[0], elem_tree);
   PT_TEST_LOG_SUCC("ProMfgprocitsmElemtreeGet");
      return status;
    }
/*--------------------------------------------------------*\
    FUNCTION : PTTestMfgProcItemVisit
    PURPOSE  : Action function for ProMfgProctableVisit
\*--------------------------------------------------------*/
ProError PTTestMfgProcItemVisit
  ( ProMfgprocItem* item, ProError error,
    ProAppData app_data)
 {  ProAnnotation annotation;
  ProMfgstepHolesetEndType end_type;
  ProDrillDepthType depth_type;
  status = ProMfgprocitemAnnotationGet (item, &annotation);
  PT_TEST_LOG_SUCC ("ProMfgprocitemAnnotationGet");
    if ( status == PRO_TK_NO_ERROR)
    {      status = ProParameterVisit (&annotation,NULL,
           PTMfgParameterVisit,"HOLE_ADJUST");
      PT_TEST_LOG_SUCC("ProParameterVisit");
    }    if ( found == PRO_B_TRUE)
        {
      status = ProParameterVisit (&annotation,NULL,PTMfgParameterVisit,
```

```
      "HOLE_DEP_1");
      PT_TEST_LOG_SUCC("ProParameterVisit");
          status = ProParameterVisit (&annotation,NULL,PTMfgParameterVisit,
            "HOLE_DEP_2");
      PT_TEST_LOG_SUCC("ProParameterVisit");
          }
    status =  ProMfgprocitemHolesetdepthtypeGet (item,&depth_type,&end_type);
  PT_TEST_LOG_SUCC("ProMfgprocitemHolesetdepthtypeGet");
    if ((end_type == PRO_MFGSTEP_HOLESETEND_ALONG_AXIS) &&
       (depth_type ==PRO_DRILL_BLIND))
        {
            status =  ProMfgprocitemHolesetdepthSet (item ,hole_dep_1);
      PT_TEST_LOG_SUCC ("ProMfgprocitemHolesetdepthSet");
    }    if (end_type ==PRO_MFGSTEP_HOLESETEND_REFERENCE)
    {      status =  ProMfgprocitemHolesetdepthSet (item ,hole_dep_2);
      PT_TEST_LOG_SUCC ("ProMfgprocitemHolesetdepthSet");
    }
  return PRO_TK_NO_ERROR;
}/*-------------------------------------------------------------*\
    FUNCTION :  PTMfgParameterVisit
    PURPOSE  :  Visit the param.
\*-------------------------------------------------------------*/
ProError PTMfgParameterVisit(ProParameter* param, ProError status,
ProAppData data)
{  ProCharName str;
  ProParamvalue value;  ProParamvalueType  type;
  short   l_val;
  ProWstringToString(str,param-id);
  if ( strcmp (str, data) == 0)
        {      status = ProParameterValueGet (param, &value);
      PT_TEST_LOG_SUCC ("ProParameterValueGet()");
            status = ProParamvalueTypeGet (&value, &type);
      PT_TEST_LOG_SUCC ("ProParamvalueTypeGet()");
            if (type ==  PRO_PARAM_BOOLEAN)
      {        status = ProParamvalueValueGet (&value, type, (void *) &l_val);
        PT_TEST_LOG_SUCC ("ProParamvalueValueGet()");
                  if (l_val == 1)
          found  = PRO_B_TRUE;
        }
      if (type ==  PRO_PARAM_DOUBLE)      {
        if ( strcmp (str,"HOLE_DEP_1") == 0)
          {            status = ProParamvalueValueGet
             (&value, type, (void *) &hole_dep_1);
            PT_TEST_LOG_SUCC ("ProParamvalueValueGet()");
          }
        if ( strcmp (str,"HOLE_DEP_2") == 0)
          {              status = ProParamvalueValueGet
            (&value, type, (void *) &hole_dep_2);
              PT_TEST_LOG_SUCC ("ProParamvalueValueGet()");
          }
      }    }
```

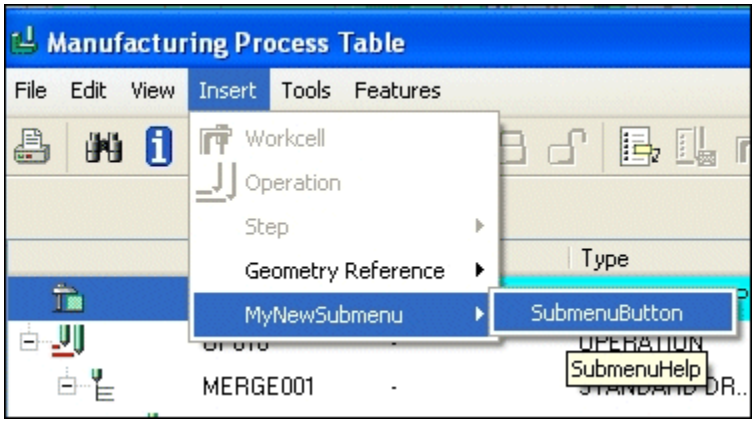
```
    return PRO_TK_NO_ERROR;
}/*---------------------------------------------------------*\
    FUNCTION :  PTMfgProcitemToolNameGet
    PURPOSE  :  Get the tool name from elem tree.
\*---------------------------------------------------------*/
ProError PTMfgProcitemToolNameGet(ProElement elem)
{  ProElempath epath;
 ProElement tool_element;  ProElempathItem  p_items[3];
 ProElemId elem_id;  ProValueDataType value_type = -1;
 ProValue     value = (ProValue)NULL;
 ProValueData value_data;  char toolname[100];
 status = ProElemtreeWrite (elem, PRO_ELEMTREE_XML, L"selecteditem_feed.inf");
  PT_TEST_LOG_SUCC (" ProElemtreeWrite ");
    status = ProElempathAlloc(&epath);
  PT_TEST_LOG_SUCC ("ProElempathAlloc ()");
    p_items[0].type = PRO_ELEM_PATH_ITEM_TYPE_ID;
  p_items[0].path_item.elem_id = PRO_E_TOOL ;
    status = ProElempathDataSet(epath, p_items, 1);
  PT_TEST_LOG_SUCC ("ProElempathDataSet()");
    status= ProElemtreeElementGet(complete_tree, epath, &tool_element);
  PT_TEST_LOG_SUCC ("ProElemtreeElementGet()");
    status = ProElementIdGet( tool_element, &elem_id);
  PT_TEST_LOG_SUCC("ProElementIdGet");
  if ( elem_id == PRO_E_TOOL )    {
      status = ProElementValueGet (tool_element, &value);
      PT_TEST_LOG_SUCC("ProElementValueGet");
            status = ProValueDataGet (value, &value_data);
      PT_TEST_LOG_SUCC("ProValueDataGet()");
            if (value_data.v.w == NULL)
         ProStringToWstring(tool_id,"TOOL_NOT_SET");
       else        ProWstringCopy(value_data.v.w,tool_id,PRO_VALUE_UNUSED);
    }  ProElempathFree(&epath);
 return PRO_TK_NO_ERROR;}
/*=====================================================================*\
  FUNCTION : user_terminate()
  PURPOSE  : To handle any termination actions
\*=====================================================================*/
void user_terminate(){
}
```

Example 4: To Add a Submenu to the Manufacturing Process Table

The following example enables you to add a menu and submenu to an existing menu option in the Manufacturing Process Table.

```
  #include <ProMenuBar.h>
  #include <ProMfgproctable.h>
  #include <ProMfg.h>
  #include <ProMdl.h>
```

```
  #include <ProMessage.h>
  #include <ProUtil.h>
  ProName submenulabel;
  ProName help;
  ProName submenubutton;
  ProError APITestCommand (ProMfg mfg, char* button_name,
             ProAppData appdata);
  ProStringToWstring (submenulabel, "MyNewSubmenu");
  ProStringToWstring (submenubutton, "SubmenuButton");
  ProStringToWstring (help, "SubmenuHelp");
  status = ProMfgproctableMenuAdd ( "APITestMenu",
  submenulabel, "Insert" );
  PT_TEST_LOG_SUCC("ProMfgproctableMenuAdd")
  status = ProMfgproctablePushbuttonAdd ("APITestMenu", "APITestMain",
                                         submenubutton,
  help, NULL, APITestCommand, NULL );
  PT_TEST_LOG_SUCC("ProMfgproctablePushbuttonAdd");
  ProError APITestCommand (ProMfg mfg, char* button_name,
ProAppData appdata) {
    printf("Just some output\n");
    return PRO_TK_NO_ERROR;
     }
```



66

Production Applications: Cabling

This chapter describes the Creo Parametric TOOLKIT cabling functions.

Cabling

This section describes the functions in Creo Parametric TOOLKIT that access the contents of a cabling harness created using the cabling module in Creo Parametric. The explanations in this section assume a knowledge of cabling and the fundamental concepts of Creo Parametric TOOLKIT, especially assemblies.

Be careful with the terminology of cabling. You can route three kinds of objects to make electrical connections within a harness: wires, cables, and bundles. The generic word for all three is cable. Unless otherwise specified, the explanations in the following sections use the word cable, in its generic sense, to include wires, cables, and bundles.

Creo Parametric TOOLKIT uses the object type `ProCable` to refer to any type of cable object. The data object has the same structure as `ProModelitem`.

```
Typedef struct pro_model_item {
     ProMdl owner;
     int id;
     protype type;
)ProCable;
```

Creating a Harness

Function Introduced:

- **ProHarnessCreate()**

`ProCable` operates on a Creo Parametric assembly. Cables can belong to one or more harness. A harness is a special Creo Parametric part designed to contain cables.

A harness cannot be retrieved into Creo Parametric Part mode. It appears in the cabling assembly as an assembly component. A harness is identified by the Creo Parametric object handle `ProHarness`.

The function `ProHarnessCreate()` creates a new harness in the specified assembly.

Finding a Harness

Function Introduced:

- **ProAssemblyHarnessesCollect()**
- **ProAssemblyHarnessesTopCollect()**
- **ProAsmcompIsHarness()**
- **ProCablingIsHarness()**
- **ProHarnessSubharnessesCollect()**

The function `ProAssemblyHarnessesCollect()` returns an array of handles to any harness that is part of a specified assembly and its sub-assemblies.

The function `ProAssemblyHarnessesTopCollect()` returns an array of handles to the harnesses present in the specified top level assembly.

Use the function `ProAsmcompIsHarness()` to identify if the specified component is a harness part. This function returns the value `TRUE`, if the component is a harness. The function `ProCablingIsHarness()` identifies if the specified model is a harness part.

Use the function `ProHarnessSubharnessesCollect()` to collect all the sub harnesses in the specified input harness part.

Finding the Cables in a Harness

Functions Introduced:

- **ProHarnessCablesCollect()**
- **ProCableHarnessesGet()**
- **ProInputFileRead()**

The function `ProHarnessCablesCollect()` provides an array of names of cables that exist in the specified harness. The output of the function includes cables that have not yet been routed.

Each cable can be created in, and routed though, several harnesses. The function `ProCableHarnessesGet()` provides an array of the handles to the harnesses below the current assembly that contain a cable with the specified name.

Use function `ProInputFileRead()` with argument `PRO_WIRELIST_TYPE` to read files in Mentor Graphics LCABLE format. This function does not create wires, but provides parameters from a wire list for use when creating in a harness assembly a wire with the same name as that in the LCABLE file.

Harness Parameters

Functions Introduced:

- **ProCablelocationAttachToHarnessComponentGet()**
- **ProCablelocationIsAttachToHarness()**

Use the function `ProCablelocationAttachToHarnessComponentGet()` to get the component on which the attach to harness location is dependent.

Use the function `ProCablelocationIsAttachToHarness()` to identify if the specified location is a cabling attach to harness location.

Importing Neutral Wire List Files

The PTC neutral wire format contains logical information for cabling. It contains information about the reference designator, pin-to-pin connection, and parameter values of connectors, pins, spools, wires, and cables.

Functions Introduced:

- **ProCablingNeutralwirelistImport()**

The function `ProCablingNeutralwirelistImport()` allows you to import and load the PTC neutral wire list file (`.nwf`) in the current session. Specify the name of the file along with its extension and full path as the input argument *filename*.

Managing Spools

Functions Introduced:

- **ProAssemblySpoolsCollect()**
- **ProSpoolCreate()**
- **ProInputFileRead()**
- **ProOutputFileMdlnameWrite()**

Function `ProAssemblySpoolsCollect()` returns a list of all spools defined in the specified assembly.

Use function `ProSpoolCreate()` to create a new spool of a given cable and sheath type in the specified assembly.

Use function `ProInputFileRead()` with argument `PRO_SPOOL_FILE` to create new spools or update existing ones. Function `ProOutputFileMdlnameWrite()` with the same argument to export a spool file.

Spool Parameters

Functions Introduced:

- **ProSpoolParameterGet()**
- **ProSpoolParametersCollect()**
- **ProSpoolParameterDelete()**
- **ProSpoolParametersSet()**
- **ProSpoolsFromLogicalGet()**
- **ProSpoolsFromLogicalCreate()**

- **ProSpoolCablesLengthGet()**
- **ProSubharnessCablesCollect()**

Function `ProSpoolParameterGet()` retrieves a single parameter for the specified spool. This function supports only single-valued parameters. If you specify a multivalued parameter, the function returns `PRO_TK_E_NOT_FOUND`.

Function `ProSpoolParameterCollect()` retrieves all parameters of the specified spool, both single- and multi-valued parameters.

Use `ProSpoolParameterDelete()` to remove a single parameter from the specified spool. This function deletes both single- and multi-valued parameters.

Function `ProSpoolParametersSet()` sets all parameters of the specified spool, both single- and multi-valued parameters. This function overwrites existing parameter values with values in the input parameter array.

Function `ProSpoolsFromLogicalGet()` returns a list of spool names in the specified assembly for which data has been imported from a logical reference but which have not yet been created. Use function `ProSpoolsFromLogicalCreate()` to create instances of spools for which logical data exists. Refer to the Creo Parametric Cabling Help and Creo Parametric Harness Help for more information on logical references.

Use the function `ProSpoolCablesLengthGet()` to get the total length of all the cables present in the specified input harness, using the specified spool.

Use the function `ProSubharnessCablesCollect()` to retrieve an array of all the cables present in the specified sub harness.

Finding Harness Connectors

Functions Introduced:

- **ProAssemblyConnectorsGet()**
- **ProConnectorsFromLogicalGet()**
- **ProAsmcompIsConnector()**

Each connector in a cabling assembly is a Creo Parametric part that is a component of that assembly. A connector can be at any level in the assembly hierarchy. Each connector is identified by its assembly component path (`ProAsmcomppath`).

The function `ProAssemblyConnectorGet()` provides an array of member identifier tables identifying the connectors in the specified assembly. The function allocates the memory for these tables.

Function `ProConnectorsFromLogicalGet()` returns a list of connector names in the specified assembly for which data has been imported from a logical reference but which have not yet been created. Refer to the Creo Parametric Cabling Help and Creo Parametric Harness Help for more information on logical references.

Use the function `ProAsmcompIsConnector()` to identify if the specified component is a cabling connector. This function returns the value `TRUE`, if the component is a cabling connector.

Connectors Parameters

Functions Introduced:

- **ProConnectorEntryPortsGet()**
- **ProConnectorParamsCollect()**
- **ProConnectorDesignate()**
- **ProConnectorWithAllParamsDesignate()**
- **ProConnectorUndesignate()**
- **ProConnectorParamGet()**
- **ProConnectorParamDelete()**
- **ProConnectorParamsSet()**
- **ProConnectorsFromLogicalGet()**
- **ProOutputFileMdlnameWrite()**
- **ProInputFileRead()**
- **ProConnectorRefModelNameGet()**

Each connector contains a set of entry ports to which cables can be connected. Each entry port is modeled by a coordinate system datum that belongs to the part that models the connector. The function `ProConnectorEntryPortsGet()` returns a `ProArray` of datum coordinate systems representing the entry ports in the specified connector. The connector is identified by its component path (its `memb_id_tab`).

The function `ProConnectorParamsCollect()` provides an array of the user parameters for the connector. However, this array contains only single-valued parameters that refer to the connector itself, not the parameters that describe the entry ports.

To access parameters on the connector entry ports, you must call the function `ProOutputFileMdlnameWrite()` with the option `PRO_CONNECTOR_PARAMS_FILE`. This writes a text file to disk, which is the same format as the file you edit when using the ProCable command **Connector**, **Modify Parameters**, **Mod Param**.

The following example shows a sample connector parameters file. Refer to the Creo Parametric Cabling Help on the parameters.

The function `ProConnectorRefModelNameGet()` retrieves the reference model name of the specified cable connector. The function returns the output argument `p_ref_model_name` as a `wchar_t*` string.

Connector Parameters File

```
! Enter or modify parameters for the connector. You may use the help
! functionality of Pro/TABLE to enter pre-defined parameters.
! Ref DescrREF_DES MOTOR
! Conn ModelMODEL_NAME      MOTOR
! Num Of PinsNUM_OF_PINS     2
! TypeTYPE    CONNECTOR
! Entry Port
!                                  TYPE    INT_LENGTH
ENTRY_PORT      ENTRY1              ROUND   0.2
ENTRY_PORT      ENTRY2              ROUND   0.2     !
Signal!
               PIN_ID  SIGNAL_NAME     SIGNAL_VALUE    ENTRY_PORT
SIGNAL          1                                      ENTRY1
SIGNAL          2
! Pin
!               PIN_ID  CABLE_NAME      COND_ID
PIN_ASSIGN      2       WIRE_1
PIN_ASSIGN      1       WIRE_2
! Enter or modify parameters for the connector. You may use the help
! functionality of Pro/TABLE to enter pre-defined parameters.
     ! Ref Descr     REF_DES MOTOR
     ! Conn Model     MODEL_NAME      MOTOR
     ! Num Of Pins     NUM_OF_PINS     2
     ! Type     TYPE    CONNECTOR
     ! Entry Port
!                      TYPE    INT_LENGTH
ENTRY_PORT      ENTRY1  ROUND   0.2
ENTRY_PORT      ENTRY2  ROUND   0.2
! Signal
!               PIN_ID  SIGNAL_NAME     SIGNAL_VALUE    ENTRY_PORT
SIGNAL          1                                      ENTRY1
SIGNAL          2
! Pin
!               PIN_ID  CABLE_NAME      COND_ID
PIN_ASSIGN      2       WIRE_1
PIN_ASSIGN      1       WIRE_2
```

Note that this file is not free-format. Each parameter name and value is followed by a tab character, and each empty value is represented by a tab character. Therefore, the line in the example that assigns the first parameter, SIGNAL,

contains three tab characters between the value of the PIN_ID and the value of ENTRY_PORT: the first tab belongs to the PIN_ID value, and next two tabs provide null values for SIGNAL_NAME and SIGNAL_VALUE.

The function `ProInputFileRead()` imports a file in this format, so you can use it in conjunction with `ProOutputFileMdlnameWrite()` to edit the parameters on connectors and their entry ports. To identify the connector, both functions use the following arguments:

- *arg1*—Represents the `memb_id_tab`
- *arg2*—Represents the `memb_num`

The function `ProConnectorDesignate()` designates a component in the assembly as a cabling connector. It takes as input the component path that identifies the part in the cabling assembly, and an optional name that will be the reference descriptor (REF_DES) of the connector.

When a new connector has been designated, it has only the two parameters REF_DES and MODEL_NAME. The MODEL_NAME is set to be the name of the part designated, and the REF_DES is set to the value provided as input to the function `ProConnectorDesignate()`, if any, or to the MODEL_NAME otherwise. After you designate a connector, you must call `ProOutputFileMdlnameWrite()` and `ProInputFileRead()` to set up the necessary parameters.

The function `ProConnectorWithAllParamsDesignate()` designates a component in the assembly as a cabling connector using all the logical parameters. The input arguments are:

- *p_connector*—Specifies the component path that identifies the part in the cabling assembly.
- *name*—Specifies the reference descriptor (REF_DES) of the connector. The argument can be NULL when the designation is not from a logical reference.
- *from_logical*—Specifies if the component must be designated using logical references.

To undesignate a connector, call the function `ProConnectorUndesignate()`.

Function `ProConnectorParamGet()` retrieves a single parameter for the specified connector. This function supports only single-valued parameters. If you specify a multivalued parameter, the function returns `PRO_TK_E_NOT_FOUND`.

Use function `ProConnectorParamDelete()` to remove a single parameter from the specified connector. This function deletes both single- and multi-valued parameters.

Function `ProConnectorParamsCollect()` retrieves all parameters of the specified connector. This function supports both single- and multi-valued parameters.

Function `ProConnectorParamsSet()` sets all parameters of the specified connector. This function overwrites all existing parameter values with the values in the input parameter array. This function supports both single- and multi-valued parameters.

Function `ProConnectorsFromLogicalGet()` returns a list of connector names in the specified assembly for which data has been imported from a logical reference but which have not yet been created. Refer to the Creo Parametric Cabling Help and Creo Parametric Harness Help for more information on logical references.

Managing Cables and Bundles

Functions Introduced:

- **ProCableCreate()**
- **ProCableAndConductorsCreate()**
- **ProBundleCreate()**
- **ProBundleCablesCollect()**
- **ProCablesegmentInfoIsInBundle()**

Use the functions `ProCableCreate()` and `ProCableAndConductorsCreate()` to create a new cable or wire in a specified harness. The type of cable created corresponds to the spool type. This function creates all required parameters with default values. If the cable or wire name has already been imported from a wire list, then parameters from that reference are used instead of default values.

Note

The function `ProCableAndConductorsCreate()` also creates conductors if needed. Pass the value of the input argument *create_conductors* as `PRO_B_TRUE` to create conductors.

Use function `ProBundleCreate()` to create a new bundle in a specified harness. The type of bundle corresponds with the spool type. This function creates all required parameters with default values.

Use the function `ProCablesegmentInfoIsInBundle()` to identify if the specified cable segment runs into a bundle. This function returns the name of the bundle only if the specified cable segment runs into a bundle.

Cable Parameters

Functions Introduced:

- **ProCableParameterGet()**
- **ProCableParameterDelete()**
- **ProCableParametersCollect()**
- **ProCableParametersSet()**
- **ProCableparammemberFree()**
- **ProCableparamproarrayFree()**
- **ProCablesFromLogicalGet()**
- **ProCablesFromLogicalCreate()**
- **ProCablesFromLogicalAllCreate()**
- **ProCableLocationsOnSegEndGet()**
- **ProOutputFileMdlnameWrite()**
- **ProInputFileRead()**
- **ProCableTessellationGet()**
- **ProCablesegmentInfoIsNew()**
- **ProCablesegmentInfoPointsGet()**
- **ProCablesegmentInfosGet()**

Function `ProCableParameterGet()` retrieves a single parameter for the specified cable. This function supports only single-valued parameters. If you specify a multivalued parameter, the function returns `PRO_TK_E_NOT_FOUND`.

Use `ProCableParameterDelete()` to remove a single parameter from the specified cable. This function deletes both single- and multi-valued parameters.

Function `ProCableParametersCollect()` retrieves all parameters of the specified cable. This function supports both single- and multi-valued parameters.

Function `ProCableParametersSet()` sets all parameters of the specified cable. This function overwrites all existing parameter values with the values in the input parameter array. This function supports both single- and multi-valued parameters.

Function `ProCableparammemberFree()` releases the memory assigned to the `ProCableparam` object. Function `ProCableparamproarrayFree()` releases the memory assigned to the array of `ProCableparam` objects. The `ProCableparam` object stores the names and values of cable parameters.

Function `ProCablesFromLogicalGet()` returns a list of cable names in the specified assembly for which data has been imported from a logical reference but which have not yet been created.

Use function `ProCablesFromLogicalCreate()` to create instances of cables for which logical data exists.

The function `ProCablesFromLogicalAllCreate()` creates cables and conductors from logical references. Cables are created using the spool features. If a cable already has a spool feature defined for it, then such cables are created using the existing spools in the model. If the spool feature is not defined for a cable, then this function also creates spools for such cables.

Refer to the Creo Parametric Cabling Help and Creo Parametric Harness Help for more information on logical references.

The function `ProCableLocationsOnSegEndGet()` returns the start and end location of each segment for the specified cable. The locations are returned as a `ProArray`.

The functions `ProOutputFileMdlnameWrite()` and `ProInputFileRead()` can be used with the option `PRO_CABLE_PARAMS_FILE` to export and import (and therefore edit) parameters on the specified cable.

Use the function `ProCableTessellationGet()` to get the tessellation for the specified input cable. Specify the following parameters as input arguments to create the surface tessellation:

- `cable`—Specify the input cable on which the tessellation is to be created.
- `input_data`—Specify the input data used for the tessellation such as `AngleControl`, `ChordHeight`, `StepSize` and so on. You can choose the options to use when generating a tessellation for the input cable.

Note

You must set the configuration option `display_thick_cables` to`yes` before using this API.

Use the function `ProCablesegmentInfoIsNew()` to identify if the specified cable segment with location information is connected to the previous segment.

Use the function `ProCablesegmentInfoPointsGet()` to get an array of points, tangents and location Ids for the specified cable segment.

Use the function `ProCablesegmentInfosGet()` to collect cable segments with location Ids of the cable segments present within the specified cable.

The following example shows a sample cable parameters file. Like the file for connector parameters, the parameter names and values are separated by tab characters.

Cable Parameters File

```
! Enter or modify parameters for the cable.
! You can use the help functionality of Pro/TABLE
! to enter pre-defined parameters.
! Cable Name
```

```
NAME    WIRE_2
! Spool Name
SPOOL   24Y
! Modify the "DIRECTION" parameter for the end type of this cable.
!               REF_DES ENTRY_PORT      DIRECTION
END_TYPE        MOTOR   ENTRY1          FROM
END_TYPE        XCONN2  ENTRY1          TO
```

Cable Identifiers and Types

Functions Introduced:

- **ProCableByNameGet()**
- **ProCableNameGet()**
- **ProCableTypeGet()**

The functions `ProCableByNameGet()` and `ProCableNameGet()` provide the handle of a cable given its name. The functions also return the cable name when supplied with the handle.

The function `ProCableTypeGet()` provides the type of a named cable. The possible types are as follows:

- A wire—A single conductor
- A cable—With several conductors
- A bundle—A collection of other wires, cables, and bundles

Cable Cosmetic Features

Functions Introduced:

- **ProCableCosmeticFeatureCreate()**
- **ProCableCosmeticFeatureTypeGet()**
- **ProCableCosmeticDistanceGet()**
- **ProCablelocationMaxDiameterGet()**
- **ProCablelocationHeightDimensionGet()**
- **ProCableTapeWindsGet()**
- **ProCableTapeWindsSet()**
- **ProCableTapeFeatSpoolGet()**
- **ProFeatureIsCableCosmetic()**
- **ProCableCosmeticfeatureEntityGet()**
- **ProCableCosmeticfeatureThicknessGet()**
- **ProCableCosmeticfeatureReferencethicknessGet()**

The function `ProCableCosmeticFeatureCreate()` creates a cabling cosmetic feature. The types of cabling cosmetic features are tie wraps, markers, and tape. The selected cable location or cable segment point to use for the feature creation. If creating a tape feature, this must contain a cable location. If creating a marker, this must contain a point on the cable segment. If creating a tie wrap, this could be a cable location or a point on a cable segment.

Use the function `ProCableCosmeticFeatureTypeGet()` to retrieve the type of the specified cosmetic feature. The enumerated data type `ProCableCosmeticType` is used to define the types of cosmetic features.

The function `ProCableCosmeticDistanceGet()` retrieves the position of a cosmetic feature relative to the start or end of a cable segment. The output arguments are listed below:

- *p_offset*—Distance of the cosmetic feature from the start or end of a segment.
- *p_start*—Boolean value which indicates if the distance is measured from the start position of a segment.

Use the function `ProCablelocationMaxDiameterGet()` to retrieve the maximum diameter of the location for the specified harness.

Use the function `ProCablelocationHeightDimensionGet()` to retrieve the height dimension for the specified cable location. The function returns the error `PRO_TK_BAD_CONTEXT` if the specified location does not have a height dimension.

The functions `ProCableTapeWindsGet()` and `ProCableTapeWindsSet()` provide access to the number of winds in a tape cosmetic feature.

The function `ProCableTapeFeatSpoolGet()` returns the spool for the specified tape feature.

Use the function `ProFeatureIsCableCosmetic()` to check if the specified feature is a cabling cosmetic feature. Cabling cosmetic feature comprises of tie wraps, markers, and tape. For more information on cabling cosmetic feature, refer to the Creo Parametric Help.

Use the function `ProCableCosmeticfeatureEntityGet()` to get the entity for the cabling cosmetic feature. This function returns a line entity for the tie wrapper and tape types of cosmetic features, and spline entity is returned if the cosmetic feature is of the a marker type.

Use the function `ProCableCosmeticfeatureThicknessGet()` to get the thickness of the specified input cable cosmetic feature.

Use the function `ProCableCosmeticfeatureReferencethicknessGet()` to obtain the thickness of the reference cable which is wrapped by the input cable cosmetic feature.

Cable Connectivity

Function Introduced:

- **ProCableLogicalEndsGet()**

The function `ProCableLogicalEndsGet()` identifies the entry ports and their owning connectors to which the specified cable should be connected. This function depends on the connector parameters SIGNAL and PIN_ASSIGN. A cable connects logically to an entry port if the connector has a PIN_ASSIGN parameter that relates a PIN_ID value to that CABLE_NAME, and a SIGNAL parameter that relates the same PIN_ID value to that ENTRY_PORT name.

In the sample connector parameters file, the cable WIRE_2 connects to connector MOTOR, entry port ENTRY1.

A cable can have logical ends, even if it has not yet been routed.

The output of the function `ProCableLogicalEndsGet()` is in the form of two `Pro/Selection` structures for the coordinate system datums that represent the entry ports.

Cable Routing Locations

Functions Introduced:

- **ProCableLocationsCollect()**
- **ProHarnessLocationsCollect()**
- **ProCablelocationTypeGet()**
- **ProCablelocationPointGet()**
- **ProCablelocationCablesGet()**

The locations though which a cable is routed are identified by `ProCableLocation` structures, which are of the same structure as `ProModelItem`. The `ProCableLocationsCollect()` function provides an array of the structures for the locations through which the specified cable is routed. The function `ProHarnessLocationsCollect()` provides an array of the structures for the locations in a specified harness.

The function `ProCablelocationTypeGet()` gives the type of a specified location. The following table lists the valid location types.

Location Type	Equivalent Command in the **CBL ROUTE** Menu
`PRO_LOC_CONNECTOR`	**Connector**
`PRO_LOC_POINT`	**Pnt/Vertex**
`PRO_LOC_FREE`	**Free**
`PRO_LOC_DEPENDENT`	**Dependent**
`PRO_LOC_AXIS`	**Along Axis**

Location Type	Equivalent Command in the CBL ROUTE Menu
PRO_LOC_USE_DIR	**Use Dir**
PRO_LOC_OFFSET	**Offset**
PRO_LOC_SPLICE	**Splice**
PRO_LOC_LOC	**Location**
PRO_LOC_OFFSET_CSYS	**Coordinate Offset**
PRO_LOC_OFFSET_AXIS	**Axis Offset**

The function `ProCablelocationPointGet()` provides the XYZ coordinates of the location in the coordinate system of the harness.

The function `ProCablelocationCablesGet()` provides an array of the names of cables routed through the specified location.

Cable Geometry

Functions Introduced:

- **ProCableLengthGet()**
- **ProCableSegmentsGet()**
- **ProCablesegmentPointsGet()**
- **ProCablesegmentIsInBundle()**
- **ProCablesegmentIsNew()**

The function `ProCableLengthGet()` provides the length of the specified wire within a specified harness.

The functions `ProCableSegmentsGet()` and `ProCablesegmentPointsGet()` provide the geometry of a named wire, bundle, or cable within a specified harness part. The geometry of a wire or cable is divided into a number of segments, each of which represents a region where the wire or cable is bundled with other wires and cables. The geometry of each such segment is described by a series of three-dimensional locations and tangent vectors.

The function `ProCablesegmentIsInBundle()` determines whether a cable segment runs into a bundle.

The function `ProCablesegmentIsNew()` determines whether the cable segment is connected to a previous cable segment.

Measuring Harness Clearance

Function Introduced:

- **ProCableClearanceCompute()**

The function `ProCableClearanceCompute()` determines the minimum distance between two items in a harness. The items can be of any of the following types, in any combination:

- Part
- Surface
- Cable
- Cable location

The inputs identify the two items in terms of `ProSelection` structures. The function outputs a flag to show whether the two items interfere, and, if they do not interfere, the function also returns the three-dimensional locations of the two nearest points.

Cable Routing

Functions Introduced:

- **ProCableRoutingStart()**
- **ProCableThruLocationRoute()**
- **ProCableRoutingEnd()**
- **ProCablelocationrefAlloc()**
- **ProCablelocationrefFree()**

To Route a Group of Cables Through a Sequence of Locations:

1. Call `ProCableRoutingStart()` to identify the cables to be routed.
2. Call `ProCablelocationrefAlloc()` to create a routing reference location structure.
3. Call `ProCableThruLocationRoute()` for each location through which to route the cables.
4. Call `ProCablelocationrefFree()` to free the location reference.
5. Call `ProCableRoutingEnd()` to complete the routing.
6. Call `ProSolidRegenerate()` to make Creo Parametric calculate the resulting cable geometry and create the necessary cable features.

> **Note**
>
> You must also call the function `ProWindowRepaint()` to see the new cables.

After the call to `ProCableRoutingStart()`, the information about the routing in progress is contained in an opaque data structure `ProRouting` that `ProCableRoutingStart()` provides. This pointer is then given as an input to the functions `ProCableThruLocationRoute()` and `ProCableRoutingEnd()`.

The inputs to `ProCableRoutingStart()` are the cabling assembly and harness handles, and an array of cables.

The input to `ProCableThruLocationRoute()` is a structure of type `ProCablelocationref`, which contains all the necessary information about the location through which to route the cables.

The following table shows the possible values of the `type` field, and the values that other fields need for each type.

type	refs	axis_flip	offsets
`PROLOC_CONNECTOR`	The coordinate system datum for the entry port	—	—
`PROLOC_POINT`	The datum point	—	—
`PROLOC_AXIS`	The axis	0 or 1 to show the routing direction	-—
`PROLOC_OFFSET`	The coordinate system datum to define the offset directions	—	Offset distances from the previous location
`PROLOC_LOC`	An existing routing location	—	—

The function `ProCableThruLocationRoute()` also outputs an array of the structures for the locations created as a result of the call. (The function usually creates a single location, but creates two in the case of routing through an axis.)

As input, the `ProCableRoutingEnd()` function takes only the *void** for the routing data.

Deleting Cable Sections

Function Introduced:

- **ProCableSectionsDelete()**

The function `ProCableSectionsDelete()` deletes the section of cables that lies between designated locations. `ProCableSectionsDelete()` does not delete loom bundle cable sections.

67

Production Applications: Piping

This chapter contains information about the Piping API functions. The functions in this section allow a Creo Parametric TOOLKIT application to create, read, and write pipe linestock information. These APIs also support analysis of pipeline connectivity as built with the Creo Parametric module Piping.

Piping Terminology

Creo Parametric TOOLKIT supports Piping. Piping uses specific terminology. This section defines this terminology.

A `pipeline` is set of interconnecting pipes and fitments. A pipeline consists of an extension which terminates at open ends (that is, ends with no further pipeline items are attached), non-open ends (that is, ends with equipment such as nozzles or other pipelines), or junctions. A pipeline also contains extensions that branch from extensions which branch from it, then others which branch from those, and so on.

A pipeline extension is a non-branching sequence of pipeline items.

A `pipeline feature` is a feature which names the pipeline to show its grouping. All other features in the pipeline refer to this feature. A pipeline feature does not contain any geometry of its own.

At a `branch` (or `junction`), pipes are grouped into extensions such that the extension which continues across the branch has a continuous direction of flow, and, if that criterion leaves a choice, has the smallest change of direction possible for that branch. Other pipes which join that branch then form the end points of other extensions.

A `member` of an extension is a terminator, a series, or a junction.

A `terminator` is the open or non-open ends of the pipeline.

A `series` is a non-branching sequence of pipeline objects.

A `junction` is an assembly component or a datum point which represents a part which joins three or more pipe segments.

A `stubin` is a datum point which joints three or more series.

A `segment` is a section of pipe, either straight or arced. If arced, the segment is manufactured by taking a straight section of tube and bending it.

A `fitting` is a component that connects two pipe segments, for example, to form a corner where space does not allow a bent pipe segment, or to represent an item such as a valve

A `pipeline object` is a segment, a fitting, or a stubin.

A `pipeline network` is a data structure which contains references to pipeline objects. The objects are structured to show their connectivity and sequence in relation to the flow.

Linestock Management Functions

This section presents functions for management of linestock.

Linestocks

Functions introduced:

- **ProAssemblyLnstksCollect()**
- **ProPipelineLnstkGet()**
- **ProPipelineLnstkSet()**
- **ProLnstkCreate()**

```
typedef struct pro_lnstk
{
  ProName            name;
  ProAssembly        owner;
} ProLnstk;
```

The function `ProAssemblyLnstksCollect()` finds all the linestocks defined for a specified assembly.

The functions `ProPipelineLnstkGet()` and `ProPipelineLnstkSet()` get and set the default linestock for a specified pipeline feature.

The function `ProLnstkCreate()` creates a new linestock in the specified assembly.

Linestock Parameters

Functions introduced:

- **ProLnstkParametersCollect()**
- **ProLnstkParametersSet()**
- **ProLnstkParameterAdd()**
- **ProLnstkParameterDelete()**

The parameters of a linestock differ from regular Creo Parametric parameters in that they may be organized hierarchically. The data structure `ProLnstkParam` contains the description of a linestock parameter and its member parameters, if any. Its declaration follows, along with those of its member types.

```
typedef enum
{
  PROLNSTKPRM_SINGLE,
  PROLNSTKPRM_MULTIPLE
} ProLnstkParamType;
typedef struct _pro_lnstk_param_memb_
{
  ProName            name;
  ProParamvalue      value;
} ProLnstkParamMemb;
typedef struct _pro_lnstk_param_
{
  ProName            name;
```

```
    ProLnstkParamType  param_type;
    union {
            ProParamvalue         value;
            ProLnstkParamMemb    *members;
          } lnstk_param_value;
  } ProLnstkParam;
```

The function `ProLnstkParametersCollect()` finds all the parameters for a specified linestock.

The function `ProLnstkParametersSet()` sets the parameters on a specified linestock to a specific list.

The function `ProLnstkParameterAdd()` adds a new parameter to the list of parameters on a specified linestock.

The function `ProLnstkParameterDelete()` deletes a named parameter from a linestock.

The linestock parameters can be set using the following enumerated types:

- `ProLnstkPipeSection`—Specifies the type of the pipe section as **Hollow** or **Solid**.
- `ProLnstkPipeShape`—Specifies the type of the pipe shape as **Flexible** or **Straight**.
- `ProLnstkPipeCrnrType`—Specifies the type of the pipe corner as **Bend**, **Fitting**, or **Miter Cut**. Corners are not set for flexible pipes.
- `ProLnstkPipeXSection`—Specifies the cross section of the pipe as **Circular** or **Rectangular**.

For round pipes the value of `ProLnstkPipeXSection` is set to `PROLNSTKPIPEXSECT_CIRCULAR`. Use the function `ProLnstkParametersCollect()` to access the values of the following pipe section parameters:

- `OD`—Outer diameter of the pipe.
- `WALL_THICKNESS`—Wall thickness of the pipe.

For rectangular pipes the value of `ProLnstkPipeXSection` is set to `PROLNSTKPIPEXSECT_RECTANGULAR`. Use the function `ProLnstkParametersCollect()` to access the values of the following pipe section parameters:

- `RECTANGULAR_HEIGHT`—Height of the rectangular pipe.
- `RECTANGULAR_WIDTH`—Width of the rectangular pipe
- `RECTANGULAR_ANGLE`—Rotate angle of the pipe solid part around its reference entity. The angle is relevant only in square pipes.
- `WALL_THICKNESS`—Wall thickness of the pipe.

Pipeline Features

The functions in this section are used to create and work with pipeline features. These functions also allow a Creo Parametric TOOLKIT application to create, read, and write line stock information.

Functions introduced:

- **ProPipelineSpecDrivenCreate()**
- **ProPipelineCreateFromXML()**
- **ProPipelineCreate()**
- **ProPipelineParametersCollect()**
- **ProPipelineParametersSet()**
- **ProPipelineParameterAdd()**
- **ProPipelineParameterDelete()**

The function `ProPipelineSpecDrivenCreate()` creates line stock and pipeline features according to the specification parameters defined in the structure `ProPipingSpecParams`. The pipeline feature is created using the newly created line stock feature. The name of the line stock feature is generated based on the specification parameters. The name of the pipeline feature is generated based on the specification parameter and the configuration option `pipeline_label_format`.

The input arguments are:

- *model*—Specifies the model where the pipeline feature must be created. The model must be Specfication-Driven, or the configuration option `piping_design_method` must be set to *spec_driven*.

  The configuration option `piping_design_method` enables you to set the design mode for the piping project. To activate the Spec-Driven design mode set the value of the configuration option to *spec_driven*. In this mode, the piping systems are created using the specified specifications. For Non Spec-Driven mode, set the value to `non_spec_driven`. In this mode, the piping systems are created manually without using project-specific data. To work in the User-Driven mode, set the value to `user_driven`. This mode enables you to switch between Spec-Driven and Non Spec-Driven piping design modes. You can convert existing assemblies to required design mode at any time in the design process.
- *spec_params*—Specifies the specification parameters. These parameters are defined in the structure `ProPipingSpecParams`. User must set the parameter values in the structure based on the values defined in the auto-selection file. Refer to the Creo Parametric Piping Help for more information on the auto-selection files.

- *Mnemonic*—Specifies the fluid or piping system. If the value is specified as `NULL`, then the default mnemonic defined in the Specification Directory file is used. The path and name of the Specification Directory file are set in the configuration option `piping_spec_dir_file`. If you pass an empty string, then no mnemonic value is assigned to the pipeline feature.
- *number*—Specifies a number which uniquely identifies the pipeline. If the value is specified as `NULL` or an empty string, then no number is assigned to the pipeline feature.
- *insulation*—Specifics the insulation for the pipeline. If the value is specified as `NULL`, then the default insulation defined in the Specification Directory file is used. The path and name of the Specification Directory file are set in the configuration option `piping_spec_dir_file`. If you pass an empty string, then the pipeline feature is created without insulation.
- *CreateSubAsm*—Specifies if the pipeline must be created as a new subassembly. The pipeline subassembly is created using the template model defined in the configuration option `pipeline_start_assembly_name`.
- *SubAsmName*—Specifies the name of the pipeline subassembly. If you pass the value of the argument as `NULL` or an empty string, then the name of the pipeline subassembly is generated based on the configuration option `pipeline_assembly_name_format`.
- *csys_reference*—Specifies a coordinate system for the placement of the pipeline subassembly. If the value is specified as `NULL`, then the coordinate system of the model is used to place the subassembly.

The function `ProPipelineCreateFromXML()` creates line stock and pipeline features according to the schematic information defined in the XML file for the specified pipeline label. The name of the line stock feature is generated based on the specification parameters. The name of the pipeline feature is generated based on the specification parameter and the configuration option `pipeline_label_format`.

The input arguments are:

- *model*—Specifies the model where the pipeline feature must be created. The model must be Specfication-Driven, or the configuration option `piping_design_method` must be set to *spec_driven*. The model must be enabled for schematic-driven modeling. The configuration option `piping_schematic_driven` must be set to *yes*.

  The configuration option `piping_schematic_driven` enables or disables the schematic-driven modeling mode for a piping project. The valid values are *yes* and *no*.
- *xml_file*—Specifies the path to the XML file which contains schematic information for pipelines.

- *pipeline_label*—Specifies the pipeline label for the XML file. The properties `SPEC`, `SIZE`, `SCHEDULE`, `MNEMONIC`, `NUMBER`, and `INSULATION` associated with a pipeline label are updated from the XML file. The property `GRADE`, that is, the material code is updated based on the other specification parameters. The property `CATEGORY` has its value set as *PIPE*.
- *insulation*—Specifics if the pipeline must be created with insulation. If the value is specified as `TRUE`, then the pipeline is created with insulation based on the parameter `INSULATION` defined in the XML file.
- *CreateSubAsm*—Specifies if the pipeline must be created as a new subassembly. The pipeline subassembly is created using the template model defined in the configuration option `pipeline_start_assembly_name`.
- *SubAsmName*—Specifies the name of the pipeline subassembly. If you pass the value of the argument as `NULL` or an empty string, then the name of the pipeline subassembly is generated based on the configuration option `pipeline_assembly_name_format`.
- *csys_reference*—Specifies a coordinate system for the placement of the pipeline subassembly. If the value is specified as `NULL`, then the coordinate system of the model is used to place the subassembly.

The function `ProPipelineCreate()` creates a Non Specification-Driven pipeline feature. The pipeline is created under the specified model. The input arguments are:

- *model*—Specifies the model where the pipeline feature must be created. The model must be Non Specfication-Driven.
- *lnstk*—Specifies the line stock feature. The line stock feature must have the specified model as its parent.
- *pipeline_name*—Specifies the name of the pipeline feature.

The function `ProPipelineParametersCollect()` retrieves all the parameters of the specified pipeline as a `ProArray`. Use the function `ProArrayFree` to release the memory assigned to the `ProArray` of parameters.

Use the function `ProPipelineParametersSet()` to set the parameters in the specified pipeline.

The function `ProPipelineParameterAdd()` adds the parameter in the specified pipeline.

The function `ProPipelineParameterDelete()` deletes the parameter in the specified pipeline.

In this section, we have explained only the configuration options which are required to work with the piping APIs. Refer to the Creo Parametric Piping Help for the complete list of piping configuration options and their detailed descriptions.

Pipeline Connectivity Analysis

The functions in the section support analysis of pipeline connectivity.

Networks

Functions introduced:

- **ProPipelineNetworkEval()**
- **ProPnetworkFree()**
- **ProPnetworkLabelGet()**
- **ProPnetworkSizeGet()**
- **ProPnetworkSpecGet()**

A pipeline is a collection of Creo Parametric piping features and components that are connected together. A pipeline feature is a single feature that unites all the features and components in a pipeline. All the features and components that belong to one pipeline reference the pipeline feature.

A network is a temporary data structure which is the result of analyzing the connectivity and topology of the features and components in a pipeline. The functions in this section allow a Creo Parametric TOOLKIT application to create and analyze the network for a pipeline, which would be the first step in, for example, an analysis of the fluid flow down the pipeline.

The network is a hierarchical data structure whose branches describe the various logical subdivisions into which the features and components of a pipeline divide themselves according to their connectivity.

A network is described by the opaque pointer ProPnetwork. The function `ProPipelineNetworkEval()` analyzes the features and components that belong to a pipeline (specified by its pipeline feature) and builds a network data structure.

After the structure has been analyzed it should be freed using `ProPnetworkFree()`.

The functions `ProPnetworkLabelGet()`, `ProPnetworkSizeGet()`, and `ProPnetworkSpecGet()` get information about the pipeline described by a specified network.

Extensions

Functions introduced

- **ProPnetworkExtensionVisit()**
- **ProPextensionFlowGet()**

A network contains a list of extensions. An extension is a non branching sequence of connected pipeline items. At a branch in a pipeline one extension is continuous across the branch and other extensions terminate at the branch. To decide which extension is continuous across the branch, the analysis performed by `ProPipelineNetworkEval()` uses the following rules:

- The extension must have a continuous direction of flow across the branch.
- Of all such possible extensions, the one chosen is the one that gives the smallest change of direction across the branch.

An extension is represented by the opaque pointer ProPextension. The function `ProPnetworkExtensionVisit()` visits all the extensions in a network.

The function `ProPextensionFlowGet()` tells you the flow direction in relation to the sequence of members in the extension.

Members

Functions introduced:

- **ProPextensionMemberVisit()**
- **ProPmemberTypeGet()**

An extension is conceptually divided into objects called members, described by the opaque object `ProPmember`. The members in an extension divide it at the pipeline branches which the extension crosses.

There are three types of member:

- Terminator—The end of a pipeline, where it either opens or connects to an item outside the pipeline, described by the opaque object `ProPterminator`.
- Junction—The item that describes how the pipeline branches, described by the opaque object `ProPjunction`.
- Series—A non branching sequence of pipeline objects, described by the opaque object `ProPseries`.

The function `ProPextensionMemberVisit()` visits all the members in an extension, and the function `ProPmemberTypeGet()` reports which of the three types the member represents. Each of three types of member is in turn composed of one or more objects.

The following sections describe the analysis of the three types of members.

Terminators

Functions introduced:

- **ProPmemberTerminatorGet()**
- **ProPterminatorTypeGet()**

The function `ProPmemberTerminatorGet()` outputs the `Pterminator` object, which represents the terminator in the specified member.

The function `ProPterminatorTypeGet()` tells you whether a terminator is an input or an output.

Junctions

Functions introduced:

- **ProPmemberJunctionGet()**

The function `ProPmemberJunctionGet()` outputs the `Pjunction` object which represents the junction in the specified member.

Series

Functions introduced:

- **ProPmemberSeriesGet()**
- **ProPseriesIdGet()**

The function `ProPmemberSeriesGet()` outputs the `Pseries` object which represents the series in the specified member.

The function `ProPseriesIdGet()` yields the integer id of the specified series.

Objects

Functions introduced:

- **ProPterminatorObjectGet()**
- **ProPjunctionObjectGet()**
- **ProPseriesObjectVisit()**
- **ProPobjectTypeGet()**
- **ProSelectionPipelineGet(**
- **ProPobjectSegmentGet()**
- **ProPobjectFittingGet()**
- **ProPfittingAsmcompGet()**
- **ProPobjectStubinGet()**
- **ProPstubinPointGet()**

- **ProPobjectSelectionGet()**
- **ProPselectionSelectionGet()**

A Piping Object describes a single item in a pipeline and is represented by the opaque pointer `Pobject`.

The functions `ProPterminatorObjectGet()` and `ProPjunctionObjectGet()` output the single object used to represent a terminator or a junction. The function `ProPseriesObjectVisit()` visits all the objects that represent the contents of a series.

The function `ProPobjectTypeGet()` yields one of the following types:

- Segment—A single pipe segment, either bent or straight. Can only belong to a series.
- Fitting—An assembly component that connects two or more pipeline segments. Can belong to a Series (if it connects two segments) or a Junction (if it connects more than two segments).
- Stubin—A datum point that defines the location where two or more pipeline segments connect directly without a fitting. Can only belong to a Junction.
- Selection—An object that contains a `ProSelection` describing the item a pipeline terminator connects to. Can only belong to a Terminator.

The function `ProSelectionPipelineGet()` outputs the pipeline feature to which the specified pipeline selection belongs.

The function `ProPobjectSegmentGet()` outputs the Segment contained by an Object of the appropriate type. The Segment is described in the next section.

The function `ProPobjectFittingGet()` outputs the fitting contained by an object of the appropriate type. The fitting is represented by the opaque object `ProPfitting`. The assembly component that represents the fitting can be found using the function `ProPfittingAsmcompGet()`.

The function `ProPobjectStubinGet()` outputs the stubin contained by an object of the appropriate type. The stubin is represented by the opaque pointer `ProPstubin`. The function `ProPstubinPointGet()` yields the 3-D location of the stubin.

The function `ProPobjectSelectionGet()` outputs the selection contained by an object of type terminator. The function `ProPselectionSelectionGet()` provides the `ProSelection` object that the selection contains and identifies the item outside the pipeline to which the terminator connects.

Segments

Functions introduced:

- **ProPsegmentTypeGet()**
- **ProPsegmentLinestockGet()**
- **ProPsegmentLengthGet()**
- **ProPsegmentCurvesGet()**

A segment is represented by the opaque pointer `ProPsegment`.

The function `ProPsegmentTypeGet()` tells you whether the segment is straight or bent.

The function `ProPsegmentLinestockGet()` outputs which linestock was used for this segment. Note that because the pipeline may contain fittings which cause a change in diameter, some segments in the pipeline may yield a different linestock from that provided by `ProPipelineLnstkGet()` for the pipeline itself.

The function `ProPsegmentLengthGet()` outputs the physical length of the segment.

The function `ProPsegmentCurvesGet()` outputs an array of `ProCurve` objects that describes the geometry of the centerline of the segment. The curves are always listed in the direction of flow.

Connecting Pipeline Segments

You can connect disconnected segments of the same pipeline or entry ports using the pipe connect feature.

The element tree for the pipe connect feature is documented in the header file `ProPipConnect.h` and is shown in the following figure:

Element Tree for Pipe Connect Feature

```
PRO_E_FEATURE_TREE
        |
        |--PRO_E_FEATURE_TYPE
        |
        |--PRO_E_STD_PIPE_LINE_ENV
        |       |--PRO_E_STD_PIPE_LINE_ID
        |       |--PRO_E_STD_PIPE_LINE_LNSTK
        |       |--PRO_E_STD_PIPE_LINE_CORNER_TYPE
        |       |--PRO_E_STD_PIPE_LINE_BEND_RAD
        |       |--PRO_E_STD_PIPE_LINE_MITER_NUM
        |       |--PRO_E_STD_PIPE_LINE_MITER_LEN
        |
        |--PRO_E_PIPE_CONNECT_FROM_MAIN_REF
        |--PRO_E_PIPE_CONNECT_TO_MAIN_REF
        |
        |--PRO_E_PIPE_CONNECT_DIMS_SCHEME
        |
        |--PRO_E_PIPE_ROUTE_ENDS
        |      |--PRO_E_PIPE_ROUTE_END_FIRST
        |            |--PRO_E_PIPE_ROUTE_END_OPT
        |            |--PRO_E_PIPE_ROUTE_END_LENGTH
        |            |--PRO_E_PIPE_ROUTE_END_REF
        |            |--PRO_E_PIPE_CONNECT_END_ANGLE
        |      |--PRO_E_PIPE_ROUTE_END_SECOND
        |            |--PRO_E_PIPE_ROUTE_END_OPT
        |            |--PRO_E_PIPE_ROUTE_END_LENGTH
        |            |--PRO_E_PIPE_ROUTE_END_REF
        |            |--PRO_E_PIPE_CONNECT_END_ANGLE
```

The following table describes the elements in the element tree for the pipe connect feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Mandatory element. Specifies the type of the feature. The valid value for this element is PRO_FEAT_PIPE_JOIN.
PRO_E_STD_PIPE_LINE_ENV	PRO_ELEM_TYPE_COMPOUND	This compound element defines the pipe options.
PRO_E_STD_PIPE_LINE_ID	PRO_ELEM_TYPE_INT	This element is mandatory, except the pipe route environment.

Element ID	Data Type	Description
		Specifies the ID of the pipeline.
PRO_E_STD_PIPE_LINE_LNSTK	PRO_ELEM_TYPE_INT	Optional element. Specifies the line stock. The default line stock is taken from the related pipeline.
PRO_E_STD_PIPE_LINE_CORNER_TYPE	PRO_ELEM_TYPE_OPTION	Optional element. Specifies the type of corner for the connect feature. The segments in the connect feature are joined using corners. The types of corner are set in the line stock. The default type of corner is taken from the related pipeline. The valid values for types of corner are: • PRO_PIPE_CORNER_TYPE_FITTING—Creates sharp corners. You can later add corner fittings. Fitted corners create breaks in a pipeline. • PRO_PIPE_CORNER_TYPE_MITER—Creates a corner by adding a miter cut. • PRO_PIPE_CORNER_TYPE_BEND—Creates each corner by bending the pipe.
PRO_E_STD_PIPE_LINE_BEND_RAD	PRO_ELEM_TYPE_DOUBLE	Optional element. This element is relevant when the corner type is set to PRO_PIPE_CORNER_TYPE_BEND. Specifies the radius of the bend. The default bend radius is taken from the related pipeline. For Specification-Driven pipelines, the value of bend radius is defined in the line stock, which is related to the pipeline.
PRO_E_STD_PIPE_LINE_MITER_NUM	PRO_ELEM_TYPE_INT	Optional element. This element is relevant when the corner type is set to PRO_PIPE_CORNER_TYPE_MITER. Specifies the number of miter cuts. The default number of cuts is taken from the related pipeline.

Element ID	Data Type	Description
		For Specification-Driven pipelines, the number of miter cuts is defined in the line stock, which is related to the pipeline.
PRO_E_STD_PIPE_LINE_MITER_LEN	PRO_ELEM_TYPE_DOUBLE	Optional element. This element is relevant when the corner type is set to PRO_PIPE_CORNER_TYPE_MITER. Specifies the length of the miter cut. The default length is taken from the related pipeline. For Specification-Driven pipelines, the length of the miter cut is defined in the line stock, which is related to the pipeline.
PRO_E_PIPE_CONNECT_FROM_MAIN_REF	PRO_ELEM_TYPE_SELECT	Mandatory element. Specifies the first end of the connect feature.
PRO_E_PIPE_CONNECT_TO_MAIN_REF	PRO_ELEM_TYPE_SELECT	Mandatory element. Specifies the second end of the connect feature.

Element ID	Data Type	Description
PRO_E_PIPE_CONNECT_DIMS_SCHEME	PRO_ELEM_TYPE_OPTION	Optional element. Specifies the dimensioning scheme to be used to connect the two ends. The valid values are: • PRO_PIPE_DIM_SCHEME_L1_L2—Sets an offset from both ends of the connect. This is the default option. • PRO_PIPE_DIM_SCHEME_L1_A1—Sets an offset from the first end of the connect and the angle between the first segment and middle segment of the connect. • PRO_PIPE_DIM_SCHEME_L1_A2—Sets an offset from the first selected end of the connect, and the angle between the middle segment and the second end. • PRO_PIPE_DIM_SCHEME_L2_A1—Sets an offset from the second selected end of the connect, and the angle between the first segment and middle segment of the connect. • PRO_PIPE_DIM_SCHEME_L2_A2—Sets an offset from the second selected end of the connect, and the angle between the middle segment and the second end.

Element ID	Data Type	Description
		• PRO_PIPE_DIM_SCHEME_A1_A2—Sets the angles between the end segments and the middle segment of the connect. **Note** If length is missing, then its value is considered as 0. If angle is missing, then the default dimensioning scheme PRO_PIPE_DIM_SCHEME_L1_L2 is used. Here again, if length is missing, then its value is considered as 0. Refer to the Creo Parametric Piping help for more information on the segments and angles created by the connect feature.
PRO_E_PIPE_ROUTE_ENDS	PRO_ELEM_TYPE_COMPOUND	This compound element defines the offset and angle values for the ends in the connect feature.

The two main elements of PRO_E_PIPE_ROUTE_ENDS are:

- PRO_E_PIPE_ROUTE_END_FIRST—This compound element specifies the values for the first end of the connect feature.
- PRO_E_PIPE_ROUTE_END_SECOND—This compound element specifies the values for the second end of the connect feature.

The following elements are common to the both the compound elements:

Element ID	Data Type	Description
PRO_E_PIPE_ROUTE_END_OPT	PRO_ELEM_TYPE_OPTION	Optional element. Specifies the type of offset. The valid values are: • PRO_PIPE_OFFSET_REFERENCE—Specifies that the offset is defined from an reference object. The reference object can be a datum plane or coordinate system, which is perpendicular to the end axis or coordinate axis. • PRO_PIPE_OFFSET_END—Specifies that the offset is defined from the selected end.
PRO_E_PIPE_ROUTE_END_LENGTH	PRO_ELEM_TYPE_DOUBLE	Optional element. Specifies the value for offset lengths. Depending on the dimensioning scheme, specify the value length_1 for L1 and length_2 for L2. The default value is 0.0.
PRO_E_PIPE_ROUTE_END_REF	PRO_ELEM_TYPE_SELECT	This element is mandatory if the type of offset is set to PRO_PIPE_OFFSET_REFERENCE. Specifies the reference object.
PRO_E_PIPE_CONNECT_END_ANGLE	PRO_ELEM_TYPE_DOUBLE	Optional element. Specifies the value for the segment angles. Depending on the dimensioning scheme, specify angle_1 for A1 and angle_2 for A2. The default value is 0.0. **Note** If the angle is missing, then the dimensioning scheme PRO_E_PIPE_CONNECT_DIMS_SCHEME is changed to PRO_PIPE_DIM_SCHEME_L1_L2. The default values of L1 and L2 are 0.0.

68

Production Applications: Welding

Welding is an optional Creo Parametric module that allows you to model welds in assemblies. In addition, you can generate report tables about weld parameters and show welding symbols in assembly drawings. This chapter provides a brief overview of weld features. For more information, refer to the “Welding Design” module in the Creo Parametric Online Help.

Read Access to Weld Features

The functions listed in this section provide access to basic information about existing weld features.

Functions introduced:

- **ProWeldTypeGet()**
- **ProWeldInfoGet()**
- **ProWeldIntermittenceGet()**
- **ProWeldSequenceIdGet()**
- **ProMdlIsSolidWeld()**
- **ProWeldGeomTypeGet()**
- **ProWeldRodGet()**
- **ProWeldRodNameGet()**
- **ProWeldCompoundGet()**
- **ProWeldFilletdataGet()**
- **ProWeldGroovedataGet()**
- **ProWeldPlugdataGet()**
- **ProWeldSlotdataGet()**
- **ProWeldSpotdataGet()**
- **ProWeldExtendedInfoToXMLExport()**

Use function `ProWeldTypeGet()` to output the type and subtype of the specified weld. Function `ProWeldInfoGet()` outputs the information you get by using the Creo Parametric command **Info**, in the **Weld** group, under the welding tab.

Function `ProWeldIntermittenceGet()` outputs information about an intermittent weld, describing the size, number, and location of the welds that form it.

Use function `ProWeldSequenceIdGet()` to obtain the sequence ID of a weld feature.

In Creo Simulate, you can add the welds created in Creo Parametric Welding application to models. During meshing, solid elements are created for solid weld objects while surface weld objects are compressed to shells. The function `ProMdlIsSolidWeld()` checks if the specified solid has been created from a solid weld.

Use the function `ProWeldGeomTypeGet()` to get the type of geometry representation for the specified weld. The types of geometry representations are defined in the enumerated data type `ProWeldGeomType`. The valid values are:

- `PRO_WELD_LIGHT`—Light welds reference existing curves or edges but have no geometry of their own. The welds are represented by the edge or surface geometry it references.
- `PRO_WELD_SURFACE`—Surface welds creates and shows the surface geometry. It is represented by surface geometry.
- `PRO_WELD_SOLID`—Solid welds and edge preparations are geometric models that offer mass properties such as volume and surface area.

Refer to the Creo Parametric Welding for more information.

Use function `ProWeldRodGet()` to provide the feature handle of the rod for the specified weld feature. Function `ProWeldRodNameGet()` gets the name of the specified weld rod feature.

`ProWeldCompoundGet()` outputs the list of welds in a compound weld.

Use functions `ProWeldFilletdataGet()`, `ProWeldGroovedataGet()`, `ProWeldPlugdataGet()`, `ProWeldSlotdataGet()`, and `ProWeldSpotdataGet()` to output data on a specific fillet, groove, plug, slot, or spot weld respectively.

Use the function `ProWeldExtendedInfoToXMLExport()` to print the information that is necessary to automatize the welding info file, in XML format.

Customizing Weld Drawing Symbols

Functions introduced:

- **ProDrawingWeldSympathGetAction()**
- **ProDrawingWeldGroupsGetAction()**
- **ProDrawingWeldSymtextGetAction()**

This section describes three notification functions invoked by Creo Parametric when the user instantiates a weld symbol that documents a weld in drawing mode. Your callback functions can output information which is used to modify the weld symbol that appears on the drawing. The effect is to allow much greater customization of the appearance of the weld symbol than is possible without Creo Parametric TOOLKIT .

Note

From Pro/ENGINEER Wildfire 5.0 onward, you can also create weld symbols in weld features as 3D Symbol Annotation Elements. Creo Parametric TOOLKIT allows you to access the Weld Symbol Annotation Elements using exisitng `ProAnnotation*()` functions. For more information on the functions, refer the Annotations: Annotation Features and Annotations on page 541 chapter.

Each callback has input arguments which identify the drawing, the weld assembly, the weld feature being annotated, and the path to the drawing symbol being used. The functions for read-access to welds, described in the previous section, would be used inside the callbacks to find out about the weld being annotated.

Refer to the Event-driven Programming: Notifications on page 2010 chapter for more data on how to set a notification.

Weld symbol notification types are:

- `PRO_DRAWING_WELD_SYMPATH_GET`—allows the callback function to override the entire weld symbol by specifying the path and file name of a substitute symbol.
- `PRO_DRAWING_WELD_GROUPIDS_GET`—allows the callback to selectively include or exclude symbol groups contained in the symbol. Additional inputs to the callback are a flag to show which way the symbol will point (left or right) and an array of the names of the groups in the symbol; the output is an array of booleans which select the groups to be included.
- `PRO_DRAWING_WELD_SYMTEXT_GET`—allows the callback to substitute for variable text in the symbol.

All three notifications can be set at the same time, allowing you to use your own set of generic symbols which are designed to be customized according to the weld type and properties.

Example 1: Weld Callback Notification

The sample code in `UgWeld.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_weld` shows how to use weld callback notification functions.

69

Creo Simulate: Items

This chapter describes how to access the properties of Creo Simulate items. The functions described in this chapter evaluate the model's structural characteristics and thermal profile and provide powerful tools for examining mechanism performance.

Entering the Creo Simulate Environment

You can access the Creo Simulate functions in one of the following situations:

- When the Creo Parametric session is in the Creo Simulate user interface for a given model.
- When the application initializes the Creo Simulate environment for a given model.

Functions Introduced:

- **ProMechanicaEnter()**
- **ProMechanicaLeave()**
- **ProMechanicaIsActive()**

The function `ProMechanicaEnter()` allows you to enter the Creo Simulate environment to access information about the Creo Simulate items in a specified model. The model must be displayed in the window.

Note

- You cannot call the function `ProMechanicaEnter()` from `user_initialize()`.
- Models created in Creo Simulate Lite mode, in both Structure and Thermal, are not supported by the Creo Parametric TOOLKIT functions. If you access a Creo Simulate Lite model in the Creo Simulate environment, the function `ProMechanicaEnter()` returns an error `PRO_TK_CANT_ACCESS`.

The function `ProMechanicaLeave()` exits the Creo Simulate environment entered using the previous function.

The function `ProMechanicaIsActive()` identifies whether the Creo Simulate environment is currently active. The environment might be active if the user has entered the Creo Simulate environment interactively, or if the function `ProMechanicaEnter()` has been called.

Note

Only functions related to the Creo Simulate database must be called between the calls to the functions `ProMechanicaEnter()` and `ProMechanicaLeave()`. `ProMechanicaEnter()` must not be used to initialize the Creo Simulate User Interface.

Entering the Creo Simulate Environment with Failed Features

You can open a model created in Creo Parametric with features that failed to regenerate in the Creo Simulate environment. The failed features appear on the Model Tree for the part.

You can use the functions described in this chapter to enter and run applications in Creo Simulate environment on a model with failed features or components. You can also create simulation objects on geometry from failed features or components and regenerate the model.

Selection of Creo Simulate Items

Creo Parametric TOOLKIT supports selection of certain Creo Simulate items. Refer to the chapter User Interface: Selection on page 503 for details about selection in Creo Parametric. These items are selected using `ProSelect()` or are obtained from the selection buffer while Creo Parametric is in the Creo Simulate Environment. The following table lists the selectable items and their `ProSelect()` filter strings:

Item Type	ProSelect() Filter String	Model Item Type
Load	sim_load*	`PRO_SIMULATION_LOAD`
Constraint	sim_load*	`PRO_SIMULATION_CONSTRAINT`
Beam	sim_beam	`PRO_SIMULATION_BEAM`
Spring	sim_spring	`PRO_SIMULATION_SPRING`
Gap	sim_gap	`PRO_SIMULATION_GAP`
Mass	sim_mass	`PRO_SIMULATION_MASS`
Shell	sim_shell	`PRO_SIMULATION_SHELL`
Shell pair	sim_shlpair	`PRO_SIMULATION_SHELL_PAIR`
Weld	sim_weld	`PRO_SIMULATION_WELD`
Interface	sim_connect	`PRO_SIMULATION_INTERFACE`

*Interactive selection using this filter will by default allow the user to select both loads and constraints. Use the `ProSelect()` filters if you desire to only allow selection of one of these types.

Example 1: Interactively Selecting and Deleting a Creo Simulate Item

The sample code in the file `PTMechExDelete.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_mech_examples/pt_mech_ex_src` shows how to select a Creo Simulate item interactively and delete it.

Accessing Creo Simulate Items

The functions described in this section allow access to Creo Simulate items in Creo Parametric.

Functions Introduced:

- **ProSolidMechitemVisit()**
- **ProMechitemNameGet()**
- **ProMechitemStatusGet()**
- **ProMechitemFilterAction()**
- **ProMechitemVisitAction()**

The Creo Simulate item is a derivative of the structure `ProModelitem` and is defined as

```
typedef struct pro_model_item
{
  ProType type;
  int id;
  ProMdl owner;
  }ProMechItem;
```

The function `ProSolidMechitemVisit()` traverses the Creo Simulate items in a specified model. This function allows you to specify the type of item to be located, or optionally all item types can be visited.

The function `ProMechitemNameGet()` returns the name of the Creo Simulate item.

The function `ProMechitemStatusGet()` returns the visibility status of the item in the current Creo Simulate environment.

The function `ProMechitemFilterAction()` is used to filter Creo Simulate items for visiting them.

The function `ProMechitemVisitAction()` is used to visit a Creo Simulate item.

Creo Simulate Object References

References to specific Creo Simulate items are contained within the Creo Simulate object references structure. This structure is represented in Creo Parametric TOOLKIT by the opaque handle, `ProMechObjectref`.

Functions Introduced:

- **ProMechobjectrefAlloc()**
- **ProMechobjectrefTypeGet()**
- **ProMechobjectrefTypeSet()**
- **ProMechobjectrefIdGet()**
- **ProMechobjectrefIdSet()**
- **ProMechobjectrefPathGet()**
- **ProMechobjectrefPathSet()**
- **ProMechobjectrefFree()**
- **ProMechobjectrefProarrayFree()**

The function `ProMechobjectrefAlloc()` allocates memory for the Creo Simulate object references handle.

The function `ProMechobjectrefTypeGet()` returns the type of the specified Creo Simulate object. The output argument *type* is one of the `PRO_SIMULATION*` types.

The function `ProMechobjectrefTypeSet()` sets the type of the Creo Simulate object.

The function `ProMechobjectrefIdGet()` obtains the ID of the specified Creo Simulate object.

The function `ProMechobjectrefIdSet()` sets the ID of the specified Creo Simulate object.

The function `ProMechobjectrefPathGet()` returns the complete path of the Creo Simulate object reference from the root assembly to the part or assembly that owns the specified Creo Simulate object.

The function `ProMechobjectrefPathSet()` sets the complete path for the Creo Simulate object reference.

Use the function `ProMechobjectrefFree()` to release the memory assigned to the Creo Simulate object reference handle.

Use the function `ProMechobjectrefProarrayFree()` to release the memory assigned to a `ProArray` of Creo Simulate object reference handles.

Geometric References

Creo Simulate items use a geometric reference structure to contain references to Creo Parametric geometry items. This structure is represented in Creo Parametric TOOLKIT by the opaque handle `ProMechGeomref`.

Functions Introduced:

- **ProMechgeomrefAlloc()**
- **ProMechgeomrefTypeGet()**
- **ProMechgeomrefSubtypeGet()**
- **ProMechgeomrefIdGet()**
- **ProMechgeomrefPathGet()**
- **ProMechgeomrefFree()**
- **ProMechgeomrefProarrayFree()**
- **ProMechgeomrefTypeSet()**
- **ProMechgeomrefSubtypeSet()**
- **ProMechgeomrefIdSet()**
- **ProMechgeomrefPathSet()**

The function `ProMechgeomrefAlloc()` allocates memory for the geometric entity. The function returns a handle to the geometric entity.

The function `ProMechgeomrefTypeGet()` returns the *type* for the specified geometric entity. The output argument *type* can have one of the following values:

- `PRO_MECH_POINT`—Specifies a point.
- `PRO_MECH_EDGE`—Specifies an edge.
- `PRO_MECH_SURFACE`—Specifies a surface.
- `PRO_MECH_VERTEX`—Specifies a vertex.
- `PRO_MECH_QUILT`—Specifies a quilt.
- `PRO_MECH_BOUNDARY`—Specifies a boundary. This type is valid for all surfaces.
- `PRO_MECH_CURVE`—Specifies a curve.
- `PRO_MECH_MODEL`—Specifies a model. This type is valid for all parts and assemblies.
- `PRO_MECH_AXIS`—Specifies the axis.
- `PRO_MECH_COORD_SYSTEM`—Specifies the coordinate system.
- `PRO_MECH_LAYER`—Specifies a layer.

- PRO_MECH_VOLUME—Specifies a set of associated surfaces that visually represents an entity with volume.
- PRO_MECH_INT*—Specifies the datum reference features that store the design intent objects. Intent objects are families of associated points, curves, edges, or surfaces that logically define boundaries of geometry created or modified by a feature. The types of datum reference features available are:
  - PRO_MECH_INT_PNT—Specifies intent datum point references.
  - PRO_MECH_INT_CURVE—Specifies intent curve references.
  - PRO_MECH_INT_EDGE—Specifies intent edge references.
  - PRO_MECH_INT_SURFACE—Specifies intent surface references.
- PRO_MECH_FEAT—Specifies the references to a Weld Feature. The weld feature should be of type:
  - Groove or Fillet
  - A surface weld
- PRO_MECH_COSMETIC—Specifies a cosmetic entity that is created as a container for lattice beams or walls. The cosmetic entities are created by simplified lattice features, and also by features that intersect or copy the lattices, such as extrude and mirror features.
- PRO_MECH_BODY—Specifies a body.

The function ProMechgeomrefTypeSet() sets the type of the geometric entity.

The function ProMechgeomrefSubtypeGet() returns the subtypes of the specified geometric entity. Only certain types of geometric entities require subtypes. The geometric entity types and their respective subtypes are as follows:

- PRO_MECH_POINT
  - PRO_MECH_POINT_SINGLE—Specifies the placement of a point at any location.
  - PRO_MECH_POINT_FEATURE—Specifies the placement of a point along a surface.
  - PRO_MECH_POINT_PATTERN—Specifies the placement of a point along a curve.
- PRO_MECH_VERTEX
  - PRO_MECH_VERTEX_EDGE_START—Specifies the start point of the referenced edge.
  - PRO_MECH_VERTEX_EDGE_END—Specifies the end point of the referenced edge.

- PRO_MECH_COORD_SYSTEM
  - PRO_MECH_CSYS_CARTESIAN—Specifies a Cartesian coordinate system.
  - PRO_MECH_CSYS_CYLINDRICAL—Specifies a cylindrical coordinate system.
  - PRO_MECH_CSYS_SPHERICAL—Specifies a spherical coordinate system.
- PRO_MECH_SURFACE
  - PRO_MECH_SURFACE_NORMAL—Specifies that the surface reference uses the standard normal direction.
  - PRO_MECH_SURFACE_REVERSED—Specifies that the surface reference uses the standard normal direction.
- PRO_MECH_CURVE
  - PRO_MECH_CURVE_NORMAL—Specifies that the curve proceeds in the default direction (from t=0 to t=1).
  - PRO_MECH_CURVE_REVERSED—Specifies that the curve reference uses the reverse direction of the curve.
- PRO_MECH_EDGE
  - PRO_MECH_EDGE_SURF_0
  - PRO_MECH_EDGE_SURF_1
- PRO_MECH_FEAT
  - PRO_MECH_FEAT_3D_LATT—Specifies a 3D lattice.
  - PRO_MECH_FEAT_2P5D_LATT—Specifies a 2.5D lattice.

The function ProMechgeomrefSubtypeSet() sets the subtypes for the specified geometric entity.

The function ProMechgeomrefIdGet() returns the ID of the specified entity.

The function ProMechgeomrefIdSet() sets the ID of the specified entity.

The function ProMechgeomrefPathGet() returns the complete path of the assembly-component references from the root assembly to the part or assembly that owns the specified geometric reference entity.

The function ProMechgeomrefPathSet() sets the complete path of the assembly-component references.

Use the function ProMechgeomrefFree() o free the geometric reference entity from the memory.

Use the function ProMechgeomrefProarrayFree() to free the array of geometric entities from the memory.

Y-directions

Several types of Creo Simulate items require a Y-direction (indicating a direction governing the properties of the item). In Creo Parametric TOOLKIT , Y-directions are represented using the opaque handle `ProMechYDirection`. The functions described in this section provide access to the y-direction handle.

Functions Introduced:

- **ProMechydirectionAlloc()**
- **ProMechydirectionTypeGet()**
- **ProMechydirectionCsysGet()**
- **ProMechydirectionCsysSet()**
- **ProMechydirectionReferenceGet()**
- **ProMechydirectionReferenceSet()**
- **ProMechydirectionVectorGet()**
- **ProMechydirectionVectorSet()**
- **ProMechydirectionFree()**

The function `ProMechydirectionAlloc()` allocates memory for the y-direction handle.

The function `ProMechydirectionTypeGet()` returns the type of the y-direction. Pass the y-direction handle as the input to this function. The output value *type* can have the following values:

- `PRO_MECH_YDIR_VECTOR`—Specifies a direction vector.
- `PRO_MECH_YDIR_REF`—Specifies a referenced coordinate system.
- `PRO_MECH_YDIR_CSYS`—Specifies a world coordinate system.

The function `ProMechydirectionCsysGet()` returns the coordinate system if the specified y-direction handle is of type `PRO_MECH_YDIR_CSYS`.

The function `ProMechydirectionCsysSet()` sets the coordinate system for the specified y-direction handle. Calling this function changes the Y-direction type to the appropriate type and discards any data related to its previous type.

The function `ProMechydirectionReferenceGet()` returns the reference entity if the specified y-direction handle is of type `PRO_MECH_YDIR_REF`.

The function `ProMechydirectionReferenceSet()` sets the reference entity for the specified y-direction handle. Calling this function changes the Y-direction type to the appropriate type and discards any data related to its previous type.

The function `ProMechydirectionVectorGet()` returns the vector direction if the specified y-direction handle is of type `PRO_MECH_YDIR_VECTOR`.

The function `ProMechydirectionVectorSet()` sets the vector for the specified y-direction handle. Calling this function changes the Y-direction type to the appropriate type and discards any data related to its previous type.

Use the function `ProMechydirectionFree()` to free the y-direction handle.

Functions

The functions described in this section provide access to the data and contents of Creo Simulate function items.

Function items use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_FUNCTION`.

Functions Introduced:

- **ProMechfuncCsysGet()**
- **ProMechfuncCsysSet()**
- **ProMechfuncDataGet()**
- **ProMechfuncDataSet()**
- **ProMechfuncDescriptionGet()**
- **ProMechfuncDescriptionSet()**
- **ProMechfuncVartypeGet()**
- **ProMechfuncVartypeSet()**
- **ProMechfuncdataTypeGet()**
- **ProMechfuncdataTypeSet()**
- **ProMechfuncdataExpressionGet()**
- **ProMechfuncdataExpressionSet()**
- **ProMechfuncdataFuncvalueinterpGet()**
- **ProMechfuncdataFuncvalueinterpSet()**
- **ProMechfuncdataIndependentvarGet()**
- **ProMechfuncdataIndepvarinterpGet()**
- **ProMechfuncdataIndepvarinterpSet()**
- **ProMechfuncdataMirrordeflectionflagGet()**
- **ProMechfuncdataMirrordeflectionflagSet()**
- **ProMechfuncdataFunctableGet()**
- **ProMechfuncdataFunctableSet()**
- **ProMechtablentryFunctionvalueGet()**
- **ProMechtablentryFunctionvalueSet()**

- **ProMechtablentryIndependentvarGet()**
- **ProMechtablentryIndependentvarSet()**
- **ProMechfuncdataAlloc()**
- **ProMechtablentryAlloc()**
- **ProMechtablentryFree()**
- **ProMechtablentryProarrayFree()**
- **ProMechfuncdataFree()**

The function `ProMechfuncCsysGet()` gets the reference co-ordinate system for the specified Creo Simulate function item. Use the function `ProMechfuncCsysSet()` to set the reference co-ordinate system for the specified Creo Simulate function item.

The function `ProMechfuncDataGet()` returns the handle to the function data of the Creo Simulate item. Use the function `ProMechfuncDataSet()` to set the value of the function data from the Creo Simulate function item

The function `ProMechfuncDescriptionGet()` returns the description of the Creo Simulate function item. Use the function `ProMechfuncDescriptionSet()` to set the function description of the Creo Simulate item

The function `ProMechfuncdataVartypeGet()` gets the function variation type of the Creo Simulate item. Use the function `ProMechfuncdataVartypeSet()` to set the function variation type for the Creo Simulate item.

The function variation types are defined by the enumerated type `ProMechfuncVarType`, which has the following values:

- `PRO_MECH_FUNC_UNIVERSAL`—Specifies the default function type for the load.
- `PRO_MECH_FUNC_COORD`—Specifies the load as a function of the current coordinate system.
- `PRO_MECH_FUNC_TIME`—Specifies the load as a function of time.
- `PRO_MECH_FUNC_TEMPERATURE`—Specifies the load as a function of temperature.
- `PRO_MECH_FUNC_DEFLECTION`—Specifies the load as a function of deflection.
- `PRO_MECH_FUNC_ARCLENGTH`—Specifies the load as a function of arc length.

- PRO_MECH_FUNC_COORDS_TIME—Specifies the load as a combination of spatial (function of current coordinate system) and temporal (function of time) functions.
- PRO_MECH_FUNC_ARCLENGTH_TIME—Specifies the load as a combination of spatial (function of arc length) and temporal (function of time) functions.

The function ProMechfuncdataTypeGet() determines the type of the function used to create the Creo Simulate item. Specify the handle to the function data as the input for this function.

The output argument *value* specifies the type of the function and can have the following values:

- PRO_MECH_FUNCTION_SYMBOLIC—Specifies a symbolic expression for a function.
- PRO_MECH_FUNCTION_TABLE—Specifies a function created using data from an interpolation table.

Use the function ProMechfuncdataTypeSet() to set the function type to be used to create the Creo Simulate item.

The function ProMechfuncdataExpressionGet() returns the symbolic expression for the specified symbolic function. Use the function ProMechfuncdataExpressionSet() to set the symbolic expression for the specified symbolic function.

The function ProMechfuncdataFuncvalueinterpGet() specifies the interpolation method used for the function value of the tabular function. The output argument *value* can have the following values:

- PRO_MECH_TABLE_LINEAR—This method linearly interpolates the variable between the values.
- PRO_MECH_TABLE_LOGARITHMIC—This method linearly interpolates the log of the variable between values.

Use the function ProMechfuncdataFuncvalueinterpSet() to set the interpolation method for the function value of the tabular function.

The function ProMechfuncdataIndependentvarGet() returns the type of the independent variable for the specified tabular function. The independent variable corresponds to the coordinate system axes and has the following values:

- PRO_MECH_INDEP_VAR_X—Specifies the value of the X-axis in the Cartesian coordinate system.
- PRO_MECH_INDEP_VAR_Y—Specifies the value of the Y-axis in Cartesian coordinate system
- PRO_MECH_INDEP_VAR_Z—Specifies the value of the Z-axis in the Cartesian or cylindrical coordinate system.

- PRO_MECH_INDEP_VAR_R—Specifies the value of the radius in a cylindrical or spherical coordinate system.
- PRO_MECH_INDEP_VAR_THETA—Specifies the value of the angle in a cylindrical or spherical coordinate system.
- PRO_MECH_INDEP_VAR_PHI—Specifies the value of the second angle in a spherical coordinate system.
- PRO_MECH_INDEP_VAR_TIME—Specifies the value of the time variable (for a time-dependent function).

Use the function ProMechfuncdataIndependentvarSet() to set the independent variable type for the specified tabular function.

The function ProMechfuncdataIndepvarinterpGet() specifies the interpolation method used for the independent variable of the tabular function. Use the function ProMechfuncdataIndepvarinterpSet() to set the interpolation method to be used for the independent variable of the tabular function.

The function ProMechfuncdataMirrordeflectionflagGet() gets the value of the mirror flag for negative deflections from the function data.

Use the function ProMechfuncdataMirrordeflectionflagSet() to set the value of the mirror flag for the negative deflections in the function data

The function ProMechfuncdataFunctableGet() returns an array of table entries for the specified tabular function. Free the array of table entries using the function ProMechtablentryProarrayFree(). Use the function ProMechfuncdataFunctableSet() to set an array of table entries for the specified tabular function.

The function ProMechtablentryFunctionvalueGet() returns the value of the specified function in the table entry. Use the function ProMechtablentryFunctionvalueSet() to set the value of the specified function in the table entry.

The function ProMechtablentryIndepedentvarGet() returns the value for the specified independent variable in the table entry. Use the function ProMechtablentryIndependentvarSet() to set the value for the specified independent variable in the table entry.

Use the function ProMechfuncdataFree() to free the array containing the function data.

Creo Simulate Expressions

Most of the ProMech*Get() functions described in the following sections return mathematical expressions of the type ProMechExpression. This mathematical expression specifies a numeric or relational value, which is

expressed and stored as a string. Use the function `ProMathExpressionEvaluate()` to evaluate such a mathematical expression. Refer to the Evaluating Mathematical Expressions for a Solid on page 106 section in the Core: Solids, Parts, and Materials on page 92 chapter for more information on this function.

Accessing the Properties used for Loads and Constraints

Loads and constraints use complicated structures to mirror the number and type of properties available in the user interface. Several structures recur in different types of loads and constraints, and are described by the following table:

Property	Creo Parametric TOOLKIT Opaque Handle
Vectored value—Specifies a value applied in a specified direction.	`ProMechvectoredvalue`
Direction vector—Represents a defined direction.	`ProMechdirvector`
Value—Specifies a scalar value potentially affected by a defined variation.	`ProMechvalue`
Variation—Specifies the components of a defined variation (either by function or by interpolation).	`ProMechvariation`
Interpolation point—Specifies an individual interpolation point used to define a variation.	`ProMechinterpolationpnt`

The types specified in the preceding table and the functions required to access these types are defined in the header file `ProMechValue.h` and `ProMechValueSet.h`.

Functions Introduced:

- **ProMechvectoredvalueAlloc**
- **ProMechvectoredvalueDirectiontypeGet()**
- **ProMechvectoredvalueDirectiontypeSet()**
- **ProMechvectoredvalueDirectionvectorGet()**
- **ProMechvectoredvalueDirectionvectorSet()**
- **ProMechvectoredvalueMagnitudeGet()**
- **ProMechvectoredvalueMagnitudeSet()**
- **ProMechvectoredvaluePointsGet()**
- **ProMechvectoredvaluePointsSet()**
- **ProMechvectoredvalueFree()**
- **ProMechdirvectorAlloc()**
- **ProMechdirvectorComponentsGet()**

- **ProMechdirvectorComponentsSet()**
- **ProMechdirvectorCsysGet()**
- **ProMechdirvectorCsysSet()**
- **ProMechdirvectorVariationGet()**
- **ProMechdirvectorVariationSet()**
- **ProMechdirvectorFree()**
- **ProMechvalueAlloc()**
- **ProMechvalueValueGet()**
- **ProMechvalueValueSet()**
- **ProMechvalueVariationGet()**
- **ProMechvalueVariationSet()**
- **ProMechvalueFree()**
- **ProMechvalueProarrayFree()**
- **ProMechvariationAlloc()**
- **ProMechvariationTypeGet()**
- **ProMechvariationFunctionidGet()**
- **ProMechvariationFunctionidSet()**
- **ProMechvariationInterpolationGet()**
- **ProMechvariationInterpolationSet()**
- **ProMechvariationFree()**
- **ProMechinterpolationpntAlloc()**
- **ProMechinterpolationpntPointGet()**
- **ProMechinterpolationpntPointSet()**
- **ProMechinterpolationpntMagnitudeGet()**
- **ProMechinterpolationpntMagnitudeSet()**
- **ProMechinterpolationpntProarrayFree()**
- **ProMechinterpolationpntFree()**
- **ProMechexternalfielddataAlloc()**
- **ProMechexternalfielddataCsysGet()**
- **ProMechexternalfielddataCsysSet()**
- **ProMechexternalfielddataFileGet()**
- **ProMechexternalfielddataFileSet()**
- **ProMechexternalfielddataFree()**

- **ProMechvariationExternalfielddataGet()**
- **ProMechvariationExternalfielddataSet()**

The function `ProMechvectoredvalueAlloc()` allocates memory for the directed value.

The function `ProMechvectoredvalueDirectiontypeGet()` returns the method used to specify the direction for the vectored value. The output argument *type* can have one of the following values:

- `PRO_MECH_DIRECTION_BY_VECTOR`—The vector is defined by specifying a direction.
- `PRO_MECH_DIRECTION_BY_2_POINTS`—The direction of the vector is specified using two points.

The function `ProMechvectoredvalueDirectiontypeSet()` sets the direction for the vectored value.

The function `ProMechvectoredvalueDirectionvectorGet()` returns the direction of vector for the vectored load of type `PRO_MECH_DIRECTION_BY_VECTOR`.

The function `ProMechvectoredvalueDirectionvectorSet()` sets the direction vector.

The function `ProMechvectoredvalueMagnitudeGet()` specifies the magnitude of the vectored load. If the value of the magnitude is positive, the load acts in the same direction as the vector and if the value of the magnitude is negative, the load acts in the direction opposite to that of the vector.

The function `ProMechvectoredvalueMagnitudeSet()` sets the magnitude of the vectored load.

The function `ProMechvectoredvaluePointsGet()` returns an array of points that define the direction of the vector if type of the vector is `PRO_MECH_DIRECTION_BY_2_POINTS`. Use the function `ProMechvectoredvaluePointsSet()` to set the array of points.

Use the function `ProMechvectoredvalueFree()` to free the memory containing the directed value.

The function `ProMechdirvectorComponentsGet()` returns the components of the vector for each coordinate direction. Use the function `ProMechdirvectorComponentsSet()` to set the component values of the vector.

The function `ProMechdirvectorCsysGet()` returns the coordinate system used to calculate the vector. Use the function `ProMechdirvectorCsysSet()` to set the coordinate system.

The function ProMechdirvectorVariationGet() specifies the spatial or time-based variation applied to the direction vector. Use the function ProMechdirvectorVariationSet() to set the spatial variation to the direction vector.

The function ProMechvalueValueGet() returns the expression used to specify the load or constraint value. Use the function ProMechvalueValueSet() to set the load or constraint value.

The function ProMechvalueVariationGet() returns the spatial or time-based variation assigned to the specified value. Use the function ProMechvalueVariationSet() to set the spatial variation.

The function ProMechvariationTypeGet() returns the variation type for the specified variation handle. The output argument *type* can have one of the following values:

- PRO_MECH_VARIATION_UNIFORM—Specifies that the variation of the load is uniform over the entity.
- PRO_MECH_VARIATION_INTERPOLATION—Specifies that the load varies along the entity as defined by the interpolation points.
- PRO_MECH_VARIATION_FUNCTION—Indicates that the variation is defined by specifying a function that is used to vary the property.

The function ProMechvariationFunctionidGet() returns the function id for the specified variation handle if the variation is of type PRO_MECH_VARIATION_FUNCTION. The function id is always extracted from the same model from where the varied value was obtained. Use the function ProMechvariationFunctionidSet() to set the function id for the variation.

The function ProMechvariationInterpolationGet() returns an array of interpolation points for the specified variation handle if the variation is of type PRO_MECH_VARIATION_INTERPOLATION. Use the function ProMechvariationInterpolationSet() to set the interpolation points for the specified variation.

The function ProMechinterpolationpntAlloc() allocates memory for the interpolation points.

The function ProMechinterpolationpntPointGet() specifies the geometric entity associated with the specified interpolation point. Use the function ProMechinterpolationpntPointSet() to set the geometric entity.

The function ProMechinterpolationpntMagnitudeGet() returns the magnitude of the load at the specified interpolation point. Use the function ProMechinterpolationpntMagnitudeSet() to set the magnitude of the load.

Use the function `ProMechinterpolationpntFree()` to free the memory containing the interpolation point. Use `ProMechinterpolationpntProarrayFree()` as a shortcut to free an entire array of interpolation points.

The function `ProMechexternalfielddataAlloc()` allocates memory for the external data structure.

The function `ProMechexternalfielddataCsysGet()` returns the reference coordinate system for the external field data. Use the function `ProMechexternalfielddataCsysSet()` to set the reference coordinate system for the external field data.

The function `ProMechexternalfielddataFileGet()` returns the information related to the external (FEM Neutral Format) FNF file to be imported for the external field. Use the function `ProMechexternalfielddataFileSet()` to set the path to the fnf file.

The function `ProMechvariationExternalfielddataGet()` returns the external field assigned for variation of a value.

Use the function `ProMechvariationExternalfielddataSet` to set the external field assigned for variation of a value.

Creo Simulate Loads

Loads are used to simulate forces that act on the model in the real world. Once you create the loads, you can examine the response of the mechanism to the loads.

Accessing Creo Simulate Loads

The functions described in this section provide a handle to the to the different types of Creo Simulate loads. Loads use the `ProType` field in the ProMechitem structure as `PRO_SIMULATION_LOAD`.

Functions Introduced:

- **ProMechloadTypeGet()**
- **ProMechloadForcedataGet()**
- **ProMechloadPressuredataGet()**
- **ProMechloadBearingdataGet()**
- **ProMechloadCentrifugaldataGet()**
- **ProMechloadGravitydataGet()**
- **ProMechloadStructtempdataGet()**
- **ProMechloadMecttempdataGet()**
- **ProMechloadHeatdataGet()**

- **ProMechloadReferencesGet()**
- **ProMechloadLoadsetsGet()**

The function `ProMechLoadTypeGet()` returns the type of load contained in the structure `ProModelItem`.

The output argument *value* can have one of the following values:

- `PRO_MECH_LOAD_FORCE`—Specifies a force load. Use the function `ProMechloadForcedataGet()` to access the data and contents of the force load structure.
- `PRO_MECH_LOAD_PRESSURE`—Specifies a pressure load. Use the function `ProMechloadPressuredataGet()` to access the data and contents of the pressure load structure.
- `PRO_MECH_LOAD_BEARING`—Specifies a bearing load. Use the function `ProMechloadBearingdataGet()` to access the data and contents of the bearing load structure.
- `PRO_MECH_LOAD_CENTRIFUGAL`—Specifies a centrifugal load. Use the function `ProMechloadCentrifugaldataGet()` to access the data and contents of the centrifugal load structure.
- `PRO_MECH_LOAD_GRAVITY`—Specifies a gravity load. Use the function `ProMechloadGravitydataGet()` to access the data and contents of the gravity load structure.
- `PRO_MECH_LOAD_STRUCTURAL_TEMPERATURE`—Specifies a temperature load. Use the function `ProMechloadStructtempdataGet()` to access the data and contents of the temperature load.
- `PRO_MECH_LOAD_MECT_TEMPERATURE`—Specifies a Mechanical temperature load. Use the function `ProMechloadMecttempdataGet()` to access the data and contents of the mechanical temperature load.
- `PRO_MECH_LOAD_HEAT`—Specifies a heat load. Use the function `ProMechloadHeatdataGet()` to access the data and contents of the heat load.

Note

From Creo Parametric onward, the load types `PRO_MECH_LOAD_GLOBAL_TEMPERATURE` and `PRO_MECH_LOAD_EXTERNAL_TEMPERATURE` have been deprecated. The functions `ProMechloadGlobaltempdataGet()` and `ProMechloadExttempdataGet()` have also been deprecated. The global temperature loads and external temperature are converted to equivalent structural temperature loads. Use the function `ProMechloadStructtempdataGet()` instead to access the data and contents of the temperature load.

The function `ProMechloadReferencesGet()` returns the geometric references. It specifies the geometric references used to define the load.

The function `ProMechloadLoadsetsGet()` returns the load set(s) that contain the given load. (Currently, Creo Simulate allows a load to be assigned to only one set).

Modifying the Creo Simulate Loads

The functions in this section assign load data to a load in the model. The load should have been created already using `ProMechitemCreate()`. Changes made via these functions will not be reflected in the user interface until you call the function `ProMechitemUpdateComplete()`.

Note

Once a load has been created and updated successfully, its load type cannot be changed.

The following is the procedure to create a new user-visible load:

1. Create the load using `ProMechitemCreate()`
2. Set the load references using `ProMechloadReferencesSet()`.
3. Set the load type-specific data using one of the `ProMechload*dataSet()` functions.
4. Assign the load to a load set using `ProMechloadLoadsetAssign()`.
5. Check the status of the load using `ProMechitemStatusGet()`.
6. Complete the load set using `ProMechitemUpdateComplete()`.

Functions Introduced:

- **ProMechloadForcedataSet()**
- **ProMechloadPressuredataSet()**
- **ProMechloadBearingdataSet()**
- **ProMechloadCentrifugaldataSet()**
- **ProMechloadGravitydataSet()**
- **ProMechloadStructtempdataSet()**
- **ProMechloadMecttempdataSet()**
- **ProMechloadHeatdataSet()**
- **ProMechloadReferencesSet()**
- **ProMechloadLoadsetAssign()**

Use the function`ProMechloadForcedataSet()` to set the data and the contents of the force load structure.

Use the function `ProMechloadPressuredataSet()` to set the data and contents of the pressure load structure.

Use the function `ProMechloadBearingdataSet()` to set the data and contents of the pressure load structure.

Use the function `ProMechloadCentrifugaldataSet()` to set the data and contents of the centrifugal load structure.

Use the function `ProMechloadGravitydataSet()` to set the data for the gravity load structure.

Use the function `ProMechloadStructtempdataSet()` to set the data of the temperature load.

Use the function `ProMechloadMecttempdataSet()` to set the data of the mechanical temperature load.

Use the function `ProMechloadHeatdataSet()` to set the data and contents of the heat load.

Use the function `ProMechloadReferencesSet()` to set the geometric references for the load. This must be a valid set of references for the load type.

Use the function `ProMechloadLoadsetAssign()` to assign a load to a particular loadset.

Note

The functions `ProMechloadGlobaltempdataSet()` and `ProMechloadExttempdataSet()` have been deprecated. From Creo Parametric onward, the global temperature loads and external temperature are converted to equivalent structural temperature loads. Use the function `ProMechloadStructtempdataSet()` instead to set the data for the temperature load.

Example 2: Modifying Magnitude of Force or Pressure Load

The sample code in the file `PTMechExMagChange.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_mech_examples/pt_mech_ex_src` shows how to modify the magnitude of a force or pressure load.

Force and Moment Loads

The functions described in this section provide read access to the data and contents of Creo Simulate force loads.

Functions Introduced:

- **ProMechforcedataAlloc()**
- **ProMechforcedataTypeGet()**
- **ProMechforcedataTypeSet()**
- **ProMechforcedataForceGet()**
- **ProMechforcedataAxialForceIgnoreflagGet()**
- **ProMechforcedataAxialForceIgnoreflagSet()**
- **ProMechforcedataForceSet()**
- **ProMechforcedataMomentGet()**
- **ProMechforcedataAxialMomentIgnoreflagGet()**
- **ProMechforcedataAxialMomentIgnoreflagSet()**
- **ProMechforcedataMomentSet()**
- **ProMechforcedataRefpntGet()**
- **ProMechforcedataRefpntSet()**
- **ProMechforcedataFree()**

The function `ProMechforcedataAlloc()` allocates memory for the Creo Simulate force load data.

The function `ProMechforcedataTypeGet()` specifies the distribution of the load over the selected entity. The output argument *value* can have one of the following values:

- `PRO_MECH_FORCE_AT_POINT`—Specifies a total load applied to a single point.
- `PRO_MECH_FORCE_TOTAL`—Specifies a load distributed along the length or area of the entity such that the integral of the load over the selected entity equals the total prescribed value.
- `PRO_MECH_FORCE_TOTAL_AT_POINT`—Specifies a Total Load At Point (TLAP) load.
- `PRO_MECH_FORCE_PER_UNIT`—Specifies a load applied to each unit that makes up the selected entity.
- `PRO_MECH_FORCE_TLAP_UNASSOCIATED`—Specifies that the Total Load At Point (TLAP) loads are un-associated from the selected entities in the model.
- `PRO_MECH_FORCE_TBLAP`—Specifies a Total Bearing Load At Point (TBLAP) for cylindrical surfaces or curves.

Use function `ProMechforcedataTypeSet()` to set the type of the force load.

The function `ProMechforcedataForceGet()` returns the magnitude and direction of the force applied to a geometric entity.

The function `ProMechforcedataAxialForceIgnoreflagGet()` returns a boolean flag indicating whether the axial force is ignored in case of a Total Bearing Load At Point (TBLAP).
`ProMechforcedataAxialForceIgnoreflagSet()` sets a boolean flag indicating whether the axial force is to be applied in case of a Total Bearing Load At Point (TBLAP).

The function `ProMechforcedataForceSet()` sets the force component for the load data.

The function `ProMechforcedataMomentGet()` returns the magnitude and direction of the moment applied to a geometric entity.

`ProMechforcedataAxialMomentIgnoreflagGet()` returns a boolean flag indicating whether the axial moment applied is ignored in case of a Total Bearing Load At Point (TBLAP).
`ProMechforcedataAxialMomentIgnoreflagSet()` sets a boolean flag indicating whether the axial moment is to be applied in case of a Total Bearing Load At Point (TBLAP).

The function `ProMechforcedataMomentSet()` sets the moment component for the load data.

The function `ProMechforcedataRefpntGet()` returns the reference point for the specified load if the load was applied at a given point. Use the function `ProMechforcedataRefpntSet()` to set the reference point.

The function `ProMechforcedataFree()` frees the memory of the force load data handle.

Pressure, Gravity, and Bearing Loads

The functions described in this section provide read access to the data and contents of the Creo Simulate pressure, gravity, and bearing loads.

Functions Introduced:

- **ProMechpressuredataAlloc()**
- **ProMechpressuredataValueGet()**
- **ProMechpressuredataValueSet()**
- **ProMechpressuredataFree()**
- **ProMechbearingdataValueGet()**
- **ProMechbearingdataValueSet()**
- **ProMechbearingdataFree()**
- **ProMechgravitydataAlloc()**
- **ProMechgravitydataValueGet()**
- **ProMechgravitydataValueSet()**
- **ProMechgravitydataFree()**

The function `ProMechpressuredataAlloc()` allocates memory for the Creo Simulate pressure load data.

The function `ProMechpressuredataValueGet()` returns the value of the pressure load.

The function `ProMechpressuredataValueSet()` sets the value of the pressure load.

The function `ProMechpressuredataFree()` frees the memory of the pressure load data handle.

The function `ProMechbearingdataValueGet()` returns the value of the bearing load.

The function `ProMechbearingdataValueSet()` sets the value of the bearing load.

The function `ProMechbearingdataFree()` frees the memory of the bearing load data handle.

The function `ProMechgravitydataAlloc()` allocates memory for the Creo Simulate gravity load data.

The function `ProMechgravitydataValueGet()` returns the value of the gravity load.

The function `ProMechgravitydataValueSet()` sets the value of the gravity load.

The function `ProMechgravitydataFree()` frees the memory of the gravity load data handle.

Centrifugal Loads

A centrifugal load results due to the rigid body rotation of the model and is applied on the entire model. The functions described in this section provide read access to the data and structure of the Creo Simulate centrifugal loads.

Functions Introduced:

- **ProMechcentrifugaldataAlloc()**
- **ProMechcentrifugaldataVelocityGet()**
- **ProMechcentrifugaldataVelocitySet()**
- **ProMechcentrifugaldataAccelerationGet()**
- **ProMechcentrifugaldataAccelerationSet()**
- **ProMechcentrifugaldataFree()**

The function `ProMechcentrifugaldataAlloc()` allocates memory for the centrifugal load data.

The function `ProMechcentrifugaldataVelocityGet()` returns the magnitude and direction of the angular velocity of the centrifugal load.

The function `ProMechcentrifugaldataVelocitySet()` sets the value of the angular velocity of the centrifugal load.

The function `ProMechcentrifugaldataAccelerationGet()` returns the acceleration vector for the centrifugal load. The acceleration vector is the rate of change of the angular velocity vector, and represents a change in magnitude or direction of the angular velocity.

The function `ProMechcentrifugaldataAccelerationSet()` sets the acceleration vector for the centrifugal load.

The function `ProMechcentrifugaldataFree()` releases the memory assigned to the centrifugal load data.

Temperature Loads

Temperature loads enable you to simulate a temperature change over your model. Temperature loads provide valuable information on how the structure of your model deforms due to a particular temperature change.

Creo Simulate Temperature Loads

The functions described in this section provide access to the data and contents of Creo Simulate temperature loads.

Functions Introduced:

- **ProMechtemperaturedataAlloc()**
- **ProMechtemperaturedataValueGet()**
- **ProMechtemperaturedataValueSet()**
- **ProMechtemperaturedataFree()**

The function `ProMechtemperaturedataAlloc()` allocates memory for the temperature load data.

The function `ProMechtemperaturedataValueGet()` returns a structure that contains the value of the temperature and the spatial variation of the load.

The function `ProMechtemperaturedataValueSet()` sets the value of the temperature and the spatial variation of the load.

The function `ProMechtemperaturedataFree()` releases the memory assigned to the temperature load data.

Structural Temperature Loads

Structural temperature loads are thermal loads resulting from a temperature change over a geometric entity or a set of geometric entities. This load is applied to curves or surfaces. If the model is an assembly, the structural load can be applied to specific assembly components.

The functions described in this section provide access to the data and contents of the Creo Simulate structural temperature loads.

Functions Introduced:

- **ProMechstructtempdataAlloc()**
- **ProMechstructtempdataValueGet()**
- **ProMechstructtempdataValueSet()**
- **ProMechstructtempdataReftempGet()**
- **ProMechstructtempdataReftempSet()**
- **ProMechstructtempdataFree()**

The function `ProMechstructtempdataAlloc()` allocates memory for the structural temperature load data.

The function `ProMechstructtempdataValueGet()` returns a structure containing the temperature applied to the load and the variation of the load.

The function `ProMechstructtempdataValueSet()` sets the value of the structural temperature data.

The function `ProMechstructtempdataReftempGet()` returns the reference temperature of the structural temperature object. The reference temperature is the stress-free temperature for the model.

The function `ProMechstructtempdataReftempSet()` sets the reference temperature of the structural temperature data.

The function `ProMechstructtempdataFree()` releases the memory assigned to structural temperature load data.

Note

From Creo Parametric onward, the global and external temperature loads are converted to equivalent structural temperature loads. The functions previously available to access Creo Simulate global and external temperature loads have been deprecated. The structural temperature load functions supersede the global and external temperature load functions.

Global Temperature Loads

A global temperature load specifies a thermal load resulting from a temperature change over the entire model.

From Creo Parametric onward, the global temperature loads are converted to equivalent structural temperature loads. The data structure `ProMechglobaltempdata` for global temperature loads in `ProMechLoad.h` has been deprecated. Functions previously available to access Creo Simulate global temperature load have also been deprecated. Use the structural temperature load data structure `ProMechstructtempdata` and the functions that access this data structure instead.For more information on structural load functions, see the section Structural Temperature Loads on page 1878.

The folowing functions have been deprecated:

- ProMechglobaltempdataAlloc()
- ProMechglobaltempdataReftempGet()
- ProMechglobaltempdataReftempSet()
- ProMechglobaltempdataValueGet()

- ProMechglobaltempdataValueSet()
- ProMechglobaltempdataFree()

MEC/T Temperature Loads

The MEC/T temperature load applies a temperature load across the entire model based on a temperature field developed from the results of a steady state or transient thermal analysis. The functions described in this section provide read and write access to the data and contents of the MEC/T temperature loads.

Functions Introduced:

- **ProMechmecttempdataAlloc()**
- **ProMechmecttempdataAnalysisidGet()**
- **ProMechmecttempdataAnalysisidSet()**
- **ProMechmecttempdataLoadsetidGet()**
- **ProMechmecttempdataLoadsetidSet()**
- **ProMechmecttempdataLoadsetSet()**
- **ProMechmecttempdataLoadsetGet()**
- **ProMechmecttempdataReftempGet()**
- **ProMechmecttempdataReftempSet()**
- **ProMechmecttempdataTimestepGet()**
- **ProMechmecttempdataTimestepSet()**
- **ProMechmecttempdataDesignstudySet()**
- **ProMechmecttempdataDesignstudyGet()**
- **ProMechmecttempdataFree()**

The function `ProMechmecttempdataAlloc()` allocates memory for the MEC/T temperature data structure.

The function `ProMechmecttempdataAnalysisidGet()` returns the ID of the thermal analysis defined for the model.

The function `ProMechmecttempdataAnalysisidSet()` sets the ID of the thermal analysis.

The function `ProMechmecttempdataLoadsetidGet()` returns the ID of the load set. The load set is applicable only if the MEC/T temperature load is coming from a steady state thermal analysis.

The function `ProMechmecttempdataLoadsetidSet()` sets the ID of the load set.

The function `ProMechmecttempdataLoadsetGet()` returns the load set in the form of the `ProMechObjectref` object for the MEC/T temperature data.

The function `ProMechmecttempdataLoadsetSet()` assigns the load set.

The function `ProMechmecttempdataReftempGet()` returns the reference temperature for the MEC/T temperature data.

The function `ProMechmecttempdataReftempSet()` sets the reference temperature.

The function `ProMechmecttempdataTimestepGet()` returns the time step value if the MEC/T temperature load is coming from a transient thermal analysis.

The function `ProMechmecttempdataTimestepSet()` sets the time step value.

The function `ProMechmecttempdataDesignstudyGet()` returns the name of the design study for the MEC/T temperature load.

The function `ProMechmecttempdataDesignstudySet()` assigns the design study name for the MEC/T temperature load.

The function `ProMechmecttempdataFree()` releases the memory assigned to the MEC/T temperature data structure.

External Temperature Loads

An external temperature load involves importing an externally calculated or measured temperature field as a temperature load. The external temperature field must contain connectivity of a linear solid element mesh, node locations, and temperature values at the nodes.

From Creo Parametric onward, the external temperature loads are converted to equivalent structural temperature loads. The data structure `ProMechexttempdata` for external temperature loads in `ProMechLoad.h` has been deprecated. Functions previously available to access Creo Simulate external temperature load have also been deprecated. Use the structural temperature load data structure `ProMechstructtempdata` and the functions that access this data structure instead. For more information on structural load functions, see the section Structural Temperature Loads on page 1878

The following functions are deprecated:

- ProMechexttempdataAlloc()
- ProMechexttempdataReftempGet()
- ProMechexttempdataReftempSet()
- ProMechexttempdataFemneutralfileGet()
- ProMechexttempdataFemneutralfileSet()
- ProMechexttempdataFree()

Heat Loads

Heat loads are entity loads and can be placed on one or more points, edgescurves, surfaces, components, or volumes. Heat loads provide local heat sources and sinks for the model and can be used to model internal heat generation or flux.

The functions described in this section provide access to the data and contents of the Creo Simulate heat loads.

Functions Introduced:

- **ProMechheatdataAlloc()**
- **ProMechheatdataTypeGet()**
- **ProMechheatdataTypeSet()**
- **ProMechheatdataValueGet()**
- **ProMechheatdataValueSet()**
- **ProMechheatdataTemporalvariationGet()**
- **ProMechheatdataTemporalvariationSet()**
- **ProMechheatdataFree()**

Functions Superseded:

- `ProMechheatdataTimefunctionidGet()`
- `ProMechheatdataTimefunctionidSet()`

The function `ProMechheatdataAlloc()` allocates memory for the heat load data.

The function `ProMechheatdataTypeGet()` returns the type of distribution for the heat load across the geometric entities. The output argument *type* can have one of the following values:

- `PRO_MECH_HEAT_TOTAL`—Specifies that the heat load is distributed along the length or area of the entity such that the integral of the load over the selected entity equals the total prescribed value.
- `PRO_MECH_HEAT_PER_UNIT`—Specifies that the heat load is applied to each unit that makes up the selected load.
- `PRO_MECH_HEAT_AT_POINT`—Specifies that the heat load is applied to a single point, a feature, or a pattern of points.
- `PRO_MECH_HEAT_NONE`—No heat load type is assigned.

The function `ProMechheatdataTypeSet()` sets the type of heat load.

The function `ProMechheatdataValueGet()` returns the total or distributed heat transfer rate, depending on the distribution option specified for the entities.

The function `ProMechheatdataValueSet()` sets the value of the heat load.

The function ProMechheatdataTemporalvariationGet() returns the time variation for the specified heat load.

ProMechheatdataTemporalvariationSet() sets the time variation for the specified heat load.

The functions ProMechheatdataTimefunctionidGet() and ProMechheatdataTimefunctionidSet() have been deprecated. Use the functions ProMechheatdataTemporalvariationGet() and ProMechheatdataTemporalvariationSet() instead.

The function ProMechheatdataFree() releases the memory assigned to the heat load data.

Creo Simulate Load Sets

A load set is a collection of loads that act together on the model. The functions described in this section provide access to data and contents of the Creo Simulate load set items. Load sets use the ProType field in the ProMechitem structure as PRO_SIMULATION_LOAD_SET.

Functions Introduced:

- **ProMechloadsetDescriptionGet()**
- **ProMechloadsetDescriptionSet()**
- **ProMechloadsetLoadsGet()**
- **ProMechloadsetTypeGet()**
- **ProMechloadLoadsetAssign()**

The function ProMechloadsetDescriptionGet() returns the name and the description of the specified load set.

The function ProMechloadsetDescriptionSet() sets the description of the specified load set.

The function ProMechloadsetLoadsGet() returns an array containing the different loads that are included in the specified load set.

The function ProMechloadsetTypeGet() returns the type of the specified load set. The type can be as follows:

- PRO_MECH_LOADSET_STRUCTURAL—Specifies a structural load set.
- PRO_MECH_LOADSET_THERMAL—Specifies a thermal load set.

Use the function ProMechloadLoadsetAssign() to assign a load to a particular loadset.

Creo Simulate Constraints

To perform analyses on the models you need to apply constraints to at least one area of the model. The constraints are associated with the model geometry and can be applied to a single geometric entity or to multiple entities.

Accessing the Creo Simulate Constraints

The functions described in this section provide access to the data and contents of the Creo Simulate constraints item. Constraints use the `ProType` field in the ProMechitem structure as `PRO_SIMULATION_CONSTRAINTS`.

Functions Introduced:

- **ProMechconstrTypeGet()**
- **ProMechconstrConvectiondataGet()**
- **ProMechconstrRadiationdataGet()**
- **ProMechconstrRadiationdataSet()**
- **ProMechconstrDisplacementdataGet()**
- **ProMechconstrSymmetrydataGet()**
- **ProMechconstrTemperaturedataGet()**
- **ProMechconstrThermalsymmetrydataGet()**
- **ProMechconstrReferencesGet()**
- **ProMechconstrConstrsetsGet()**

The function `ProMechconstrTypeGet()` returns the type of constraint contained in the structure `ProModelItem`. The output can have one of the following values:

- `PRO_MECH_CONSTR_CONVECTION`—Specifies a linear convective heat exchange condition for one or more geometric or model entities for thermal mode. Use the function `ProMechconstrConvectiondataGet()` to access the data and contents of this constraint type.
- `PRO_MECH_CONSTR_DISPLACEMENT`—Specifies a displacement constraint for structural mode. Use the function `ProMechconstrDisplacementdataGet()` to access the data and contents of this constraint type.
- `PRO_MECH_CONSTR_SYMMETRY`—Specifies cyclic symmetry for structural mode. Use the function `ProMechconstrSymmetrydataGet()` to access the data and contents of this constraint type.
- `PRO_MECH_CONSTR_RADIATION`—Specifies a thermal radiation exchange between the model surface and the surroundings. Use the function

ProMechconstrRadiationdataGet() to access the data and contents of radiation constraint type from the Creo Simulate item. The function ProMechconstrRadiationdataSet() sets the handle to the data and contents of radiation constraint type from the Creo Simulate item.

- PRO_MECH_CONSTR_TEMPERATURE—Specifies a temperature boundary condition for one or more geometric or model entities for thermal mode. Use the function to() to access the data and contents of this constraint type.
- PRO_MECH_CONSTR_SYMMETRY_THERM—Specifies a cyclic symmetry thermal constraint for thermal mode. Use the function ProMechconstrThermalsymmetrydataGet() to access the data and contents of this constraint type.
- PRO_MECH_CONSTR_INIT_TEMP—Specifies the initial temperature boundary condition for one or more geometric entities for thermal mode.

The function ProMechconstrReferencesGet() returns the geometric references. It specifies the geometric references used to define the constraint.

The function ProMechconstrConstrsetsGet() returns the constraint set (s) that contain the given constraint. Currently, Creo Simulate allows a constraint to be assigned to only one set.

Modifying the Creo Simulate Constraints

The functions in this section assign load data to a constraint in the model. The load should have been already created using ProMechitemCreate(). The changes made using these functions will not be reflected in the user interface until you call ProMechitemUpdateComplete().

Note

Once a constraint has been created and updated successfully, its constraint type cannot be changed.

The following is the procedure to create a new user-visible constraint:

1. Create the load using ProMechitemCreate().
2. Set the constraint references using ProMechconstrReferencesSet().
3. Set the constraint type-specific data using one of the ProMechconstr*dataSet() functions.
4. Assign the constraint to a constraint set using ProMechconstrConstrsetAssign().

5. Check the status of the constraint using `ProMechitemStatusGet()`.
6. Complete the constraint using `ProMechitemUpdateComplete()`.

Functions Introduced:

- **ProMechconstrConvectiondataSet()**
- **ProMechconstrDisplacementdataSet()**
- **ProMechconstrSymmetrydataSet()**
- **ProMechconstrTemperaturedataSet()**
- **ProMechconstrThermalsymmetrydataSet()**
- **ProMechconstrReferencesSet()**
- **ProMechconstrConstrsetAssign()**

The function `ProMechconstrConvectiondataSet()` sets the convection constraint data.

The function `ProMechconstrDisplacementdataSet()` sets the displacement constraint data.

The function `ProMechconstrRadiationdataSet()` sets the value for the radiation constraint data.

The function `ProMechconstrSymmetrydataSet()` sets the value of the symmetry constraint data.

The function `ProMechconstrTemperaturedataSet()` sets the value of the temperature constraint data.

The function `ProMechconstrThermalsymmetrydataSet()` sets the value of the thermal symmetry constraint data.

The function `ProMechconstrReferencesSet()` sets the value of the constraint geometric references. This must be a valid set of references for the constraint type.

Use the function `ProMechconstrConstrsetAssign()` to assign a constraint to a constraint set.

Convection Constraints

Convection constraints specify a boundary condition on the convective heat exchange between a moving fluid and geometric entities and/or element entities within your model.

The functions described in this section provide access to the data and contents of the Creo Simulate convection constraints. You can define the value of the convection constraint as a function of temperature.

Functions Introduced:

- **ProMechconvectiondataAlloc()**
- **ProMechconvectiondataBulktempGet()**
- **ProMechconvectiondataBulktempSet()**
- **ProMechconvectiondataBulktempUnset()**
- **ProMechconvectiondataFilmcoefficientGet()**
- **ProMechconvectiondataFilmcoefficientSet()**
- **ProMechconvectiondataTemperaturedependenceGet()**
- **ProMechconvectiondataTemperaturedependenceSet()**
- **ProMechconvectiondataTemporalvariationGet()**
- **ProMechconvectiondataTemporalvariationSet()**
- **ProMechconvectiondataFree()**

The function `ProMechconvectiondataAlloc()` allocates memory for the convection load data.

The function `ProMechconvectiondataBulktempGet()` returns the value of the bulk temperature. The bulk temperature specifies the temperature of the fluid in contact with the surface, during convective heat transfer through a surface.

The function `ProMechconvectiondataBulktempSet()` sets the value of the bulk temperature.

The function `ProMechconvectiondataFilmcoefficientGet()` returns the value of the film coefficient. The film coefficient specifies the constant of proportionality between the flux through the surface and the difference between the surface temperature and the bulk temperature, in convective heat transfer through a surface.

Use the function `ProMechconvectiondataFilmcoefficientSet()` to set the value of the film coefficient.

The function `ProMechconvectiondataTemperaturedependenceGet()` returns the temperature variation of the convection constraint. The convection constraint is specified as a function of temperature. Use the function `ProMechconvectiondataTemperaturedependenceSet()` to set the temperature variation for the convection constraint.

The function `ProMechconvectiondataTemporalvariationGet()` returns the temporal variation for the convection constraint. Use the function `ProMechconvectiondataTemporalvariationSet()` to set the temporal variation of the convection constraint. The temporal variation specifies whether the convection condition is in steady state or a function of time

Note

The function `ProMechconvectiondataTimefunctionidGet()` and `ProMechconvectiondataTimefunctionidSet()` have been deprecated. Use the functions `ProMechconvectiondataTemporalvariationGet()` and `ProMechconvectiondataTemporalvariationSet()` instead

The function `ProMechconvectiondataFree()` releases the memory assigned to the convection load data.

Radiation Constraints

A thermal radiation exchanges heat between the model surface and the surroundings. Radiation does not take place between the model surfaces. You can define the value of the radiation constraint as a function of temperature.

The functions described in this section provide access to the data and contents of the Creo Simulate radiation constraints.

Functions Introduced:

- **ProMechradiationdataAlloc()**
- **ProMechradiationdataAmbienttempExprGet()**
- **ProMechradiationdataAmbienttempExprSet()**
- **ProMechradiationdataEmissivityGet()**
- **ProMechradiationdataEmissivitySet()**
- **ProMechradiationdataTemperaturedependenceGet()**
- **ProMechradiationdataTemperaturedependenceSet()**
- **ProMechradiationdataFree()**

The function `ProMechradiationdataAlloc()` allocates memory for the radiation load data.

The model emits and absorbs energy from the surroundings at a fixed ambient temperature. Use the function `ProMechradiationdataAmbienttempExprGet()` to get the value of the ambient temperature for the radiation constraint data. Use the function `ProMechradiationdataAmbienttempExprSet()` to set the value of the ambient temperature for the radiation constraint data.

The function `ProMechradiationdataEmissivityGet()` gets the emissivity value. Use the function `ProMechradiationdataEmissivitySet()` to set the emissivity value.

The function `ProMechradiationdataTemperaturedependenceGet()` returns the temperature variation of the radiation constraint. Use the function `ProMechradiationdataTemperaturedependenceSet()` to set the temperature variation of the radiation constraint.

The function `ProMechradiationdataFree()` releases the memory assigned to the radiation load data.

Displacement Constraints

The functions described in this section provide access to the data and contents of the Creo Simulate displacement constraints.

Functions Introduced:

- **ProMechdisplacementdataAlloc()**
- **ProMechdisplacementdataTypeGet()**
- **ProMechdisplacementdataTypeSet()**
- **ProMechdisplacementdataCsysGet()**
- **ProMechdisplacementdataCsysSet()**
- **ProMechdisplacementdataRotationconstrsGet()**
- **ProMechdisplacementdataRotationconstrsSet()**
- **ProMechdisplacementdataTranslationconstrsGet()**
- **ProMechdisplacementdataTranslationconstrsSet()**
- **ProMechdisplacementdataFree()**
- **ProMechdisplacementregularconstrAlloc()**
- **ProMechdisplacementregularconstrTypeSet()**
- **ProMechdisplacementregularconstrTypeGet()**
- **ProMechdisplacementregularconstrValueSet()**
- **ProMechdisplacementregularconstrValueGet()**
- **ProMechdisplacementdataTranslationinterpretinradiansflagGet()**
- **ProMechdisplacementdataTranslationinterpretinradiansflagSet()**
- **ProMechdisplacementregularconstrFree()**
- **ProMechdisplacementregularconstrProarrayFree()**
- **ProMechdisplacementdataPinconstrSet()**
- **ProMechdisplacementdataPinconstrGet()**
- **ProMechdisplacementpinconstrAlloc()**
- **ProMechdisplacementdataPinangularconstrTypeSet()**

- **ProMechdisplacementdataPinangularconstrTypeGet()**
- **ProMechdisplacementdataPinaxialconstrTypeSet()**
- **ProMechdisplacementdataPinaxialconstrTypeGet()**
- **ProMechdisplacementpinconstrFree()**

The function `ProMechdisplacementdataAlloc()` allocates the memory for the displacement constraint data handle.

The function `ProMechdisplacementdataTypeGet()` returns the type of displacement constraint data. The types of displacement constraints are:

- `PRO_MECH_DISPLACEMENT_REGULAR`—Specifies an external limit on the movement of a portion of the model.
- `PRO_MECH_DISPLACEMENT_PLANE`—This constraint type allows full planar movement, but constrains any off-plane displacement.
- `PRO_MECH_DISPLACEMENT_PIN`— Creates a constraint along a cylindrical surface for 3D models.
- `PRO_MECH_DISPLACEMENT_BALL`— Creates a constraint along a spherical surface for 3D models.

Use the function `ProMechdisplacementdataTypeSet()` to set the displacement constraint data.

The function `ProMechdisplacementdataCsysGet()` returns the reference coordinate system for the displacement constraint. Use the function `ProMechdisplacementdataCsysSet()` to set the reference coordinate system for the displacement constraint.

The function `ProMechdisplacementdataRotationconstrsGet()` returns the rotational component of the displacement about the X, Y, and Z axis. Use the function `ProMechdisplacementdataRotationconstrsSet()` to set the rotational component of displacement.

The function `ProMechdisplacementdataTranslationconstrsGet()` returns the translational component of the displacement about the X, Y, and Z directions. Use the function `ProMechdisplacementdataTranslationconstrsSet()` to set the translational component of displacement.

The function `ProMechdisplacementregularconstrAlloc()` allocates memory for the regular displacement constraint data structure.

The method `ProMechdisplacementregularconstrTypeGet()` returns the type of setting for the displacement constraint. Valid values are:

- PRO_MECH_DISPLACEMENT_FREE—Allows freedom of movement in the specified direction.
- PRO_MECH_DISPLACEMENT_FIXED—Constrains the entity, preventing movement in the specified direction.
- PRO_MECH_DISPLACEMENT_ENFORCED—Specifies an enforced displacement or rotation in the specified direction.

The function ProMechdisplacementregularconstrTypeSet() sets the type of the displacement constraint.

The function ProMechdisplacementregularconstrValueGet() returns the variation settings of the displacement if the type of displacement constraint is PRO_MECH_DISPLACEMENT_ENFORCED. The variation settings are as follows:

- An enforced displacement value in length units for the translational component
- An enforced rotation in radians for the rotational component

Use the function ProMechdisplacementregularconstrValueSet() to set the variation settings of the displacement.

The function ProMechdisplacementdataTranslationinterpretinradians flagGet() returns the value true if the angular translations are interpreted in radians. This is applicable only if the displacement is of type PRO_MECH_DISPLACEMENT_REGULAR and if cylindrical or spherical coordinate system is selected for the translations. Use the function ProMechdisplacementdataTranslationinterpretinradians flagSet() to set the value of the flag interpret angular translations in radians.

The function ProMechdisplacementregularconstrFree() releases the memory assigned to the regular displacement constraint data handle.

The function ProMechdisplacementregularconstrProarrayFree() releases the memory assigned to an array of regular displacement contraints.

The function ProMechdisplacementpinconstrAlloc() allocates memory for the pin constraint data structure.

The function ProMechdisplacementdataPinconstrGet() returns the pin constraint data structure. The pin constraint can have the following properties:

- Angular—Allows you to control the rotation about the axis of the selected cylindrical surface.
- Axial—Allows you to control translation along the axis of the selected cylindrical surface.

The function ProMechdisplacementdataPinconstrSet() sets the value for the pin constraint data structure.

The function `ProMechdisplacementdataPinangularconstrTypeGet()` returns the angular constraint type for the pin constraint. Use the function `ProMechdisplacementdataPinangularconstrTypeSet()` to set the angular constraint type. Valid values are:

- `PRO_MECH_DISPLACEMENT_FREE`—Allows freedom of movement in the specified direction.
- `PRO_MECH_DISPLACEMENT_FIXED`—Constrains the entity, preventing movement in the specified direction.

Note

The angular constraint cannot be of type `PRO_MECH_DISPLACEMENT_ENFORCED`.

The function `ProMechdisplacementdataPinaxialconstrTypeGet()` returns the axial constraint type for the pin constraint.

Use the method `ProMechdisplacementdataPinaxialconstrTypeSet()` to set the axial constraint for the pin constraint. Valid values are:

- `PRO_MECH_DISPLACEMENT_FREE`—Allows freedom of movement in the specified direction.
- `PRO_MECH_DISPLACEMENT_FIXED`—Constrains the entity, preventing movement in the specified direction.

Note

The axial constraint cannot be of type `PRO_MECH_DISPLACEMENT_ENFORCED`.

Use the function `ProMechdisplacementpinconstrFree()` to free the memory containing the pin constraint data structure.

Example 3: Copying and Assigning a Displacement Constraint to a New Reference

The sample code in the file `PTMechExCopy.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_mech_examples/pt_mech_ex_src` shows how to copy a displacement constraint and assign it to a new reference.

Symmetry Constraints

A symmetry constraint allows you to analyze a section of a symmetric model that simulates the behavior of the whole part or assembly to which it belongs.

The functions described in this section provide read and write access to the data and contents of the Creo Simulate symmetry constraints.

Functions Introduced:

- **ProMechsymmetrydataAlloc()**
- **ProMechsymmetrydataTypeGet()**
- **ProMechsymmetrydataTypeSet()**
- **ProMechsymmetrydataAxisGet()**
- **ProMechsymmetrydataAxisSet()**
- **ProMechsymmetrydataSide1Get()**
- **ProMechsymmetrydataSide1Set()**
- **ProMechsymmetrydataSide2Get()**
- **ProMechsymmetrydataSide2Set()**
- **ProMechsymmetrydataFree()**

The function `ProMechsymmetrydataAlloc()` allocates the memory for the symmetry constraint data handle.

The function `ProMechsymmetrydataTypeGet()` returns the type of symmetry constraint. The argument *type* can have the following values:

- `PRO_MECH_SYMMETRY_CYCLIC`—Specifies a cyclic symmetry
- `PRO_MECH_SYMMETRY_MIRROR`—Specifies a mirror symmetry

Use the function `ProMechsymmetrydataTypeSet()` to set the type of symmetry constraint.

The function `ProMechsymmetrydataAxisGet()` returns the axis of symmetry of the section. Use the function `ProMechsymmetrydataAxisSet()` to set the axis of symmetry.

The function `ProMechsymmetrydataSide1Get()` returns the first side of the cut section. Use the function `ProMechsymmetrydataSide1Set()` to set the geometric references of the first side.

The function `ProMechsymmetrydataSide2Get()` returns the second side of the cut section. Use the function `ProMechsymmetrydataSide2Set()` to set the geometric references of the second side.

The function `ProMechsymmetrydataFree()` releases the memory assigned to the symmetry constraint data handle.

Creo Simulate Constraint Sets

A constraint set is a collection of constraints that act together, and at the same time, on the model. Constraint sets do not contain loads.

The functions referring to the Creo Simulate constraint sets use the structure `ProModelItem`. Constraint sets use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_CONSTR_SET`.

Functions Introduced:

- **ProMechconstrsetTypeGet()**
- **ProMechconstrsetDescriptionGet()**
- **ProMechconstrsetDescriptionSet()**
- **ProMechconstrsetConstrsGet()**
- **ProMechconstrConstrsetAssign()**

The function `ProMechconstrsetTypeGet()` returns the type of the constraint sets that are applied to the model. The types are as follows:

- `PRO_MECH_LOADSET_STRUCTURAL`—Specifies a structural constraint set.
- `PRO_MECH_LOADSET_THERMAL`—Specifies a thermal constraint set.

The function `ProMechconstrsetDescriptionGet()` returns the description of the constraint set. The function `ProMechconstrsetDescriptionSet()` enables you to change the description of the constraint set.

The function `ProMechconstrsetConstrsGet()` returns the constraints specified in the constraint set.

Use the function `ProMechconstrConstrsetAssign()` to assign a constraint to a constraint set.

Creo Simulate Matrix Functions

The functions described in this section provide read and write access to the matrix data of the Creo Simulate mass properties, spring properties, beam orientation, and beam section items.

Functions Introduced:

- **ProMechMatrixAlloc()**
- **ProMechMatrixComponentGet()**
- **ProMechMatrixComponentSet()**
- **ProMechMatrixFree()**

The function `ProMechMatrixAlloc()` allocates memory for the Creo Simulate matrix handle.

The function `ProMechMatrixComponentGet()` gets the individual matrix component value at the given index values. The function `ProMathExpressionEvaluate()` is used to evaluate the expression.

The function `ProMechMatrixComponentSet()` sets the individual matrix component value at the given index values. You can also set an expression in the specified index value.

The function `ProMechMatrixFree()` releases the memory assigned for the Creo Simulate matrix handle.

Creo Simulate Vector Functions

The functions described in this section provide read and write access to the vector data of the Creo Simulate a mass properties, spring properties, beam orientation, and beam section items.

Functions Introduced:

- **ProMechVectorAlloc()**
- **ProMechVectorComponentGet()**
- **ProMechVectorComponentSet()**
- **ProMechVectorFree()**

The function `ProMechVectorAlloc()` allocates memory for the Creo Simulate vector handle.

The function `ProMechVectorComponentGet()` gets the individual vector component value at the given index values. The function `ProMathExpressionEvaluate()` is used to evaluate the expression.

The function `ProMechVectorComponentSet()` sets the individual vector component value at the given index values. You can also set an expression in the specified index value.

The function `ProMechVectorFree()` releases the memory assigned for the Creo Simulate vector handle.

Creo Simulate Beams

A beam is a one-dimensional idealization that, in three dimensions, represents a structure whose length is much greater than its other two dimensions.

The functions described in this section provide read and write access to the data and contents of Creo Simulate beam items. Beams use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_BEAM`.

Functions Introduced:

- **ProMechbeamdataAlloc()**
- **ProMechbeamTypeGet()**
- **ProMechbeamMaterialIdGet()**
- **ProMechbeamMaterialIdSet()**
- **ProMechbeamReferencesGet()**
- **ProMechbeamReferencesSet()**
- **ProMechbeamBeamdataGet()**
- **ProMechbeamBeamdataSet()**
- **ProMechbeamTrussdataGet()**
- **ProMechbeamTrussdataSet()**
- **ProMechbeamsideAlloc()**
- **ProMechbeamdataCompcrvrelGet()**
- **ProMechbeamdataCompcrvrelSet()**
- **ProMechbeamdataSidesGet()**
- **ProMechbeamdataSidesSet()**
- **ProMechbeamsideOrientGet()**
- **ProMechbeamsideOrientSet()**
- **ProMechbeamsideReleaseGet()**
- **ProMechbeamsideReleaseSet()**
- **ProMechbeamsideSectionGet()**
- **ProMechbeamsideSectionSet()**
- **ProMechbeamsideFree()**
- **ProMechbeamsideProarrayFree()**
- **ProMechbeamdataStressrecoveryGet()**
- **ProMechbeamdataStressrecoverySet()**
- **ProMechbeamdataXyshearGet()**
- **ProMechbeamdataXyshearSet()**
- **ProMechbeamdataXzshearGet()**
- **ProMechbeamdataXzshearSet()**
- **ProMechbeamdataYdirectionGet()**
- **ProMechbeamdataYdirectionSet()**
- **ProMechbeamdataFree()**

- **ProMechtrussdataAlloc()**
- **ProMechtrussdataSectionGet()**
- **ProMechtrussdataSectionSet()**
- **ProMechtrussdataFree()**

The function `ProMechbeamdataAlloc()` allocates the memory for the beam data handle.

The function `ProMechbeamTypeGet()` returns the types of beams.

The different types of beams are as follows:

- `PRO_MECH_BEAM_BEAM`—Specifies a beam. Use the function `ProMechbeamBeamdataGet()` to access the data structure for the beam. Use the function `ProMechbeamBeamdataSet()` to modify the beam.
- `PROMECH_BEAM_TRUSS`—Specifies a truss. Use the function `ProMechbeamTrussdataGet()` to access the data structure for the truss. Use the function `ProMechbeamTrussdataSet()` to modify the truss.

The function `ProMechbeamMaterialIdGet()` returns the ID of the Creo Parametric material used to create the beam. Use the function `ProMechbeamMaterialIdSet()` to set the material ID for the beam.

The function `ProMechbeamReferencesGet()` returns the start and end points of the beam or a curve or surface of the beam. Use the function `ProMechbeamReferencesSet()` to set the beam references.

The function `ProMechbeamsideAlloc()` allocates the memory for the beam side data handle.

The function `ProMechbeamdataCompcrvrelGet()` returns a boolean that indicates whether the beam releases are applied only to the ends of the selected composite curve or to the ends of the individual curves. Use the function `ProMechbeamdataCompcrvrelSet()` to set whether the beam releases are to be applied only to the ends of the selected composite curve or to the ends of the individual curves.

The function `ProMechbeamdataSidesGet()` returns a structure containing the cross section properties for beam elements, the orientation properties for angle and offsets, and the degrees of freedom at each beam end. Use the function `ProMechbeamdataSidesSet()` to set the cross section properties and orientation properties of the beam data.

The function `ProMechbeamsideOrientGet()` returns the orientation ID for the specified beam elements. Use the function `ProMechbeamsideOrientSet()` to set the orientation ID.

The function `ProMechbeamsideReleaseGet()` returns the beam release ID for the specified beam elements. Use the function `ProMechbeamsideReleaseSet()` to set the beam release ID.

The function `ProMechbeamsideSectionGet()` returns the cross section ID for the specified beam elements. Use the function `ProMechbeamsideSectionSet()` to set the cross section ID.

The function `ProMechbeamsideFree()` releases the memory assigned to the beam side handle.

The function `ProMechbeamsideProarrayFree()` releases the memory assigned to a `ProArray` of beam side handles.

The function `ProMechbeamdataStressrecoveryGet()` returns a boolean that indicates whether stress recovery is included while creating the beam. If this option is specified as `true`, you can access the beam stress at a specific location on the beam section. Use the function `ProMechbeamdataStressrecoverySet()` to set the value of the boolean.

The function `ProMechbeamdataXyshearGet()` returns the shear relief coefficient due to taper for the XY planes. Use the function `ProMechbeamdataXyshearSet()` to set the shear relief coefficient relative to the XY plane.

The function `ProMechbeamdataXzshearGet()` returns the shear relief coefficient due to taper for the XZ planes. Use the function `ProMechbeamdataXzshearSet()` to set the shear relief coefficient relative to the XZ plane.

The function `ProMechbeamdataYdirectionGet()` returns the orientation of the XY plane of a beam by defining its Y-direction. Use the function `ProMechbeamdataYdirectionSet()` to set the Y-direction of the beam.

The function `ProMechbeamdataFree()` releases the memory assigned to the beam data handle.

The function `ProMechtrussdataAlloc()` allocates the memory for the truss data handle.

The function `ProMechtrussdataSectionGet()` returns the cross section ID for the specified truss elements. Use the function `ProMechtrussdataSectionSet()` to set the beam section ID for the truss data.

The function `ProMechtrussdataFree()` releases the memory assigned to the truss data handle.

Creo Simulate Beams: Sections, Sketched Sections, and General Sections

The functions described in this section provide read and write access to the data and contents of the Creo Simulate beam sections, sketched beam sections, and general beam sections items.

Functions Introduced:

- **ProMechBeamsectionTypeGet()**
- **ProMechBeamsectionTypeSet()**
- **ProMechBeamsectionIntegerGet()**
- **ProMechBeamsectionIntegerSet()**
- **ProMechBeamsectionExpressionGet()**
- **ProMechBeamsectionExpressionSet()**
- **ProMechBeamsectionMatrixGet()**
- **ProMechBeamsectionMatrixSet()**
- **ProMechBeamsectionVectorGet()**
- **ProMechBeamsectionVectorSet()**

These functions use the enumerated type `ProMechBeamsectionPropertyType` to define the beam section property type.

```
typedef enum
{
  PRO_MECH_BEAMSECTION_SKETCHED_FEATURE_ID       = 0,
  /* int */
  PRO_MECH_BEAMSECTION_SKETCHED_SHEARCENTER      = 1,
  /* ProMechVector (2) [Dy, Dz] */
  PRO_MECH_BEAMSECTION_SKETCHED_ORIENTTYPE       = 2,
  /* ProMechSketchedSectionOrient (int) */

  PRO_MECH_BEAMSECTION_SQUARE_DIMENSION          = 3,
  /* ProMechExpression [a] */

  PRO_MECH_BEAMSECTION_RECTANGLE_DIMENSION       = 4,
  /* ProMechVector (2) [b, d] */

  PRO_MECH_BEAMSECTION_HOLLOWRECTANGLE_DIMENSION = 5,
  /* ProMechVector (4) [b, d, bi, di] */

  PRO_MECH_BEAMSECTION_CHANNEL_DIMENSION         = 6,
  /* ProMechVector (4) [b, t, di, tw] */
  PRO_MECH_BEAMSECTION_CHANNEL_SHEARFACTOR       = 7,
  /* ProMechVector (2) [Fy, Fz] */

  PRO_MECH_BEAMSECTION_IBEAM_DIMENSION           = 8,
  /* ProMechVector (4) [b, t, di, tw] */

  PRO_MECH_BEAMSECTION_LSECTION_DIMENSION        = 9,
  /* ProMechVector (4) [b, t, di, tw] */
  PRO_MECH_BEAMSECTION_LSECTION_SHEARFACTOR      = 10,
  /* ProMechVector (2) [Fy, Fz] */
```

```
PRO_MECH_BEAMSECTION_DIAMOND_DIMENSION         = 11,
/* ProMechVector (2) [b, d] */

PRO_MECH_BEAMSECTION_SOLIDCIRCLE_DIMENSION     = 12,
/* ProMechExpression [R] */

PRO_MECH_BEAMSECTION_HOLLOWCIRCLE_DIMENSION    = 13,
/* ProMechVector (2) [R, Ri] */

PRO_MECH_BEAMSECTION_SOLIDELLIPSE_DIMENSION    = 14,
/* ProMechVector (2) [a, b]*/

PRO_MECH_BEAMSECTION_HOLLOWELLIPSE_DIMENSION   = 15,
/* ProMechVector (3) [a, b, ai] */
PRO_MECH_BEAMSECTION_HOLLOWELLIPSE_SHEARFACTOR = 16,
/* ProMechVector (2) [Fy, Fz] */

PRO_MECH_BEAMSECTION_GENERAL_AREA              = 17,
/* ProMechExpression [Area] */
PRO_MECH_BEAMSECTION_GENERAL_INERTIA           = 18,
/* ProMechMatrix (2x2, symmetrical)
                                                    [ Ixx  Ixy ]
                                                    [      Iyy ]
                                                 */
PRO_MECH_BEAMSECTION_GENERAL_TORSIONSTIFFNESS  = 19,
/* ProMechExpression [j] */
PRO_MECH_BEAMSECTION_GENERAL_SHEARFACTOR       = 20,
/* ProMechVector (2) [Fy, Fz] */
PRO_MECH_BEAMSECTION_GENERAL_SHEARCENTER       = 21,
/* ProMechVector (2) [Dy, Dz] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_1           = 22,
/* ProMechVector (2) [y1, z1] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_2           = 23,
/* ProMechVector (2) [y2, z2] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_3           = 24,
/* ProMechVector (2) [y3, z3] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_4           = 25,
/* ProMechVector (2) [y4, z4] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_5           = 26,
/* ProMechVector (2) [y5, z5] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_6           = 27,
/* ProMechVector (2) [y6, z6] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_7           = 28,
/* ProMechVector (2) [y7, z7] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_8           = 29,
/* ProMechVector (2) [y8, z8] */
PRO_MECH_BEAMSECTION_GENERAL_POINT_9           = 30,
/* ProMechVector (2) [y9, z9] */

PRO_MECH_BEAMSECTION_WARPCOEFFICIENT           = 31,
/* ProMechExpression
```

```
  PRO_MECH_BEAMSECTION_NONSTRUCTMASS              = 32,
  /* ProMechExpression [Non-Str Mass] */
  PRO_MECH_BEAMSECTION_NONSTRUCTMASSMOMENT        = 33,
  /* ProMechExpression [Non-Str Mass Moment] */
  PRO_MECH_BEAMSECTION_CENTERGRAVITY              = 34
  /* ProMechVector (2) [cg:y, cg:z] */
}
ProMechBeamsectionPropertyType;
```

The function `ProMechBeamsectionTypeGet()` returns the type of beam section. Use the function `ProMechBeamsectionTypeSet()` to set the type of beam section.

The types of beam sections are as follows:

- `PRO_MECH_BEAM_SECTION_SKETCHED`—Specifies a cross section created using either the sketch solid beam or sketch thin beam type. Use the enumerated type `ProMechBeamsectionPropertyType` to get and set the properties of the sketched beam section.
- `PRO_MECH_BEAM_SECTION_SQUARE`—Specifies a square beam section type. The cross section dimension is specified by the length of the sides of the square. Use the function `ProMechBeamsectionExpressionGet()` to get the square beam section data. Use the function `ProMechBeamsectionExpressionSet()` to set the properties of the square beam section.
- `PRO_MECH_BEAM_SECTION_RECTANGLE`—Specifies a rectangular beam section type. The cross section dimension is specified by the height and width of the rectangle. Use the function `ProMechBeamsectionVectorGet()` to get the rectangular beam section data. Use the function `ProMechBeamsectionVectorSet()` to set the properties of the rectangular beam section
- `PRO_MECH_BEAM_SECTION_HOLLOW_RECTANGLE`—Specifies a hollow rectangular beam section type. The cross section dimension for this beam type is specified by the Outer height and width, and the inner height and width of the rectangle. Use the function `ProMechBeamsectionVectorGet()` to get the hollow beam section data. Use the function `ProMechBeamsectionVectorSet()` to set the properties of the hollow beam section.
- `PRO_MECH_BEAM_SECTION_CHANNEL`—Specifies a channel beam section type. The cross section dimension is specified by the flange width, flange thickness, web height, and web thickness. Use the function `ProMechBeamsectionVectorGet()` to get the channel section data. Use the function `ProMechBeamsectionVectorSet()` to set the properties of the channel beam section.

- PRO_MECH_BEAM_SECTION_I_BEAM—Specifies an L-section beam section type. The cross section dimension for this beam type is specified by the overall flange width, flange thickness, web height, and web thickness. Use the function ProMechBeamsectionVectorGet() to get the L-section data. Use the function ProMechBeamsectionVectorSet() to set the properties of the L-section beam type.
- PRO_MECH_BEAM_SECTION_L_SECTION—Specifies an L-section beam section type. The cross section dimension for this beam type is specified by the overall flange width, flange thickness, web height, and web thickness. Use the function ProMechBeamsectionVectorGet() to get the L-section data. Use the function ProMechBeamsectionVectorSet() to set the properties of the L-section beam type.
- PRO_MECH_BEAM_SECTION_DIAMOND—Specifies a diamond beam section type. The cross section dimension is specified by the width and height of the sides. Use the function ProMechBeamsectionVectorGet() to get the diamond beam section data. Use the function ProMechBeamsectionVectorSet() to set the properties of the diamond beam section.
- PRO_MECH_BEAM_SECTION_SOLID_CIRCLE—Specifies a solid circle beam section type. The cross section dimension for this beam type is specified by the radius of the circular beam cross-section. Use the function ProMechsectiondataCirclesectdataGet() to access the solid circle beam section data structure. Use the function ProMechsectiondataCirclesectdataSet() to modify the solid circle beam section data structure.
- PRO_MECH_BEAM_SECTION_HOLLOW_CIRCLE—Specifies a hollow circle beam section type. The cross section dimension for this beam type is specified by the outside radius and the inside radius of the hollow beam cross-section. Use the function ProMechBeamsectionVectorGet() to get the hollow circle beam section data. Use the ProMechBeamsectionVectorSet() to set the properties of the hollow circle beam section.
- PRO_MECH_BEAM_SECTION_SOLID_ELLIPSE—Specifies a solid ellipse beam section type. The cross section dimension for this beam type is specified by the length of the major axis and the length of the minor axis. Use the function ProMechBeamsectionVectorGet() to get the solid ellipse beam data. Use the function ProMechBeamsectionVectorSet() to set the properties of the solid ellipse beam section.
- PRO_MECH_BEAM_SECTION_HOLLOW_ELLIPSE—Specifies a hollow ellipse beam section type. The cross section dimension for this beam type is specified by the length of the major axis, length of the minor axis, and the

inside major axis. Use the function `ProMechBeamsectionVectorGet()` to get the hollow ellipse beam data. Use the function `ProMechBeamsectionVectorSet()` to set the properties of the hollow ellipse beam section

- `PRO_MECH_BEAM_SECTION_GENERAL`—Specifies a general beam section type. A general beam section type does not have a predefined shape. Use the enumerated type `ProMechBeamsectionPropertyType` to get and set the properties of the general beam data.

The function `ProMechBeamsectionIntegerGet()` returns the integer value of the specified beam type property. Use the function `ProMechBeamsectionIntegerSet()` to set an integer value for the specified beam type. For example, you can get and set the beam sketch feature id, the type of orientation used for the sketched beam section, and so on.

The function `ProMechBeamsectionExpressionGet()` returns the expression of type `ProMechExpression` for the specified beam section type. To evaluate the expression use the function `ProMathExpressionEvaluate()`. The function `ProMechBeamsectionExpressionSet()` sets the expression of type `ProMechExpression` for the specified beam section type. For example, you can get and set the section area, torsion stiffness (J parameter) of the beam section, and so on.

The function `ProMechBeamsectionExpressionGet()` returns the expression of type `ProMechExpression` for the specified beam section type. To evaluate the expression use the function `ProMathExpressionEvaluate()`. The function `ProMechBeamsectionExpressionSet()` sets the expression of type `ProMechExpression` for the specified beam section type. For example, you can get and set the section area, torsion stiffness (J parameter) of the beam section, and so on.

The function `ProMechBeamsectionMatrixGet()` returns the Creo Simulate matrix handle `ProMechMatrix`. Use the function `ProMechBeamsectionMatrixSet()` to set the handle to `ProMechMatrix`. The functions `ProMechMatrixComponentGet/Set` provide read and write access to the individual matrix components of the specified beam section property type. For example, you can get/set the values for moments of intertia Ixx, Ixy, and Izz. See the section Creo Simulate Matrix Functions on page 1894, for more information on handling matrix components.

The function `ProMechBeamsectionMatrixGet()` automatically allocates memory for the Creo Parametric Creo Simulate matrix handle. Use the function `ProMechMatrixFree()` to free the assigned memory.

The function `ProMechBeamsectionVectorGet()` returns the Creo Simulate vector handle `ProMechVector`. Use the function `ProMechBeamsectionVectorSet()` to set the handle to `ProMechVector`. The functions `ProMechVectorComponentGet/Set` provide read and write access to the individual vector components of the specified beam section property type. For example, you can get/set the values for circular, square, channel, I-beam sections, and so on. See the section Creo Simulate Vector Functions on page 1895, for more information on handling vector components.

The function `ProMechBeamsectionVectorGet()` automatically allocates the memory for the Creo Simulate vector handle. Use the function `ProMechVectorFree()` to free the memory.

Creo Simulate Beam Sections

The functions described in this section provide read and write access to the data and contents of the Creo Simulate beam section items. Beam sections use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_BEAM_SECTION`.

Functions Introduced:

- **ProMechsectiondataAlloc()**
- **ProMechbeamsectionDataGet()**
- **ProMechbeamsectionDataSet()**
- **ProMechbeamsectionDescriptionGet()**
- **ProMechbeamsectionDescriptionSet()**
- **ProMechsectiondataFree()**

The beam section functions listed under Functions Superseded have been deprecated. Use the functions described in the section Creo Simulate Beams: Sections, Sketched Sections, and General Sections on page 1898 instead.

Functions Superseded:

- **ProMechsectiondataSectiontypeGet()**
- **ProMechsectiondataNonstructcogGet()**
- **ProMechsectiondataNonstructcogSet()**
- **ProMechsectiondataNonstructmassGet()**
- **ProMechsectiondataNonstructmassSet()**
- **ProMechsectiondataNonstructmomentGet()**
- **ProMechsectiondataNonstructmomentSet()**
- **ProMechsectiondataWarpcoeffGet()**

- **ProMechsectiondataWarpcoeffSet()**
- **ProMechsectiondataChannelsectdataGet()**
- **ProMechsectiondataChannelsectdataSet()**
- **ProMechsectiondataCirclesectdataGet()**
- **ProMechsectiondataCirclesectdataSet()**
- **ProMechsectiondataDiamondsectdataGet()**
- **ProMechsectiondataDiamondsectdataSet()**
- **ProMechsectiondataEllipsesectdataGet()**
- **ProMechsectiondataEllipsesectdataSet()**
- **ProMechsectiondataGeneralsectdataGet()**
- **ProMechsectiondataGeneralsectdataSet()**
- **ProMechsectiondataHollowcirclesectdataGet()**
- **ProMechsectiondataHollowcirclesectdataSet()**
- **ProMechsectiondataHollowellipsesectdataGet()**
- **ProMechsectiondataHollowellipsesectdataSet()**
- **ProMechsectiondataHollowrectsectdataGet()**
- **ProMechsectiondataHollowrectsectdataSet()**
- **ProMechsectiondataIbeamsectdataGet()**
- **ProMechsectiondataIbeamsectdataSet()**
- **ProMechsectiondataLsectionsectdataGet()**
- **ProMechsectiondataLsectionsectdataSet()**
- **ProMechsectiondataRectanglesectdataGet()**
- **ProMechsectiondataRectanglesectdataSet()**
- **ProMechsectiondataSketchedsectdataGet()**
- **ProMechsectiondataSketchedsectdataSet()**
- **ProMechsectiondataSquaresectdataGet()**
- **ProMechsectiondataSquaresectdataSet()**

The function `ProMechsectiondataAlloc()` allocates the memory for the beam section data handle.

The function `ProMechbeamsectionDataGet()` provides access to the data structure containing the properties of the beam section data. Use the function `ProMechbeamsectionDataSet()` to set the properties of the beam section data.

The function `ProMechsectiondataSectiontypeGet()` returns the types of beam sections.

The types of beam sections are as follows:

- `PRO_MECH_BEAM_SECTION_SKETCHED`—Specifies a cross section created using either the sketch solid beam or sketch thin beam type. Use the function `ProMechsectiondataSketchedsectdataGet()` to access the sketched beam data structure. Use the function `ProMechsectiondataSketchedsectdataSet()` to set the properties of the sketched beam data structure.
- `PRO_MECH_BEAM_SECTION_SQUARE`—Specifies a square beam section type. The cross section dimension is specified by the length of the sides of the square. Use the function `ProMechsectiondataSquaresectdataGet()` to access the square section data structure. Use the function `ProMechsectiondataSquaresectdataSet()` to set the properties of the square section data structure.
- `PRO_MECH_BEAM_SECTION_RECTANGLE`—Specifies a rectangular beam section type. The cross section dimension is specified by the height and width of the rectangle. Use the function `ProMechsectiondataRectanglesectdataGet()` to access the rectangular section data structure. Use the function `ProMechsectiondataRectanglesectdataSet()` to access the rectangular section data structure.
- `PRO_MECH_BEAM_SECTION_HOLLOW_RECTANGLE`— Specifies a hollow rectangular beam section type. The cross section dimension for this beam type is specified by the Outer height and width, and the inner height and width of the rectangle. Use the function `ProMechsectiondataHollowrectsectdataGet()` to access the hollow section data structure. Use the function `ProMechsectiondataHollowrectsectdataSet()` to access the hollow section data structure.
- `PRO_MECH_BEAM_SECTION_CHANNEL`—Specifies a channel beam section type. The cross section dimension is specified by the flange width, flange thickness, web height, and web thickness. Use the function `ProMechsectiondataChannelsectdataGet()` to access the channel section data structure. Use the function `ProMechsectiondataChannelsectdataSet()` to access the channel section data structure.
- `PRO_MECH_BEAM_SECTION_I_BEAM`—Specifies an I-beam section type. The cross section dimension for this beam type is specified by the flange width, flange thickness, web height, and web thickness. Use the function `ProMechsectiondataIbeamsectdataGet()` to access the I-beam section data structure. Use the function

ProMechsectiondataIbeamsectdataSet() to access the I-beam section data structure.

- PRO_MECH_BEAM_SECTION_L_SECTION—Specifies an L-section beam section type. The cross section dimension for this beam type is specified by the overall flange width, flange thickness, web height, and web thickness. Use the function ProMechsectiondataLsectionsectdataGet() to access the L-section data structure. Use the function ProMechsectiondataLsectionsectdataSet() to access the L-section data structure.
- PRO_MECH_BEAM_SECTION_DIAMOND—Specifies a diamond beam section type. The cross section dimension is specified by the width and height of the sides. Use the function ProMechsectiondataDiamondsectdataGet() to access the diamond beam section data structure. Use the function ProMechsectiondataDiamondsectdataSet() to modify the diamond beam section data structure.
- PRO_MECH_BEAM_SECTION_SOLID_CIRCLE—Specifies a solid circle beam section type. The cross section dimension for this beam type is specified by the radius of the circular beam cross-section. Use the function ProMechsectiondataCirclesectdataGet() to access the solid circle beam section data structure. Use the function ProMechsectiondataCirclesectdataSet() to modify the solid circle beam section data structure.
- PRO_MECH_BEAM_SECTION_HOLLOW_CIRCLE—Specifies a hollow circle beam section type. The cross section dimension for this beam type is specified by the outside radius and the inside radius of the hollow beam cross-section. Use the function ProMechsectiondataHollowcirclesectdataGet() to access the hollow circle beam section data structure. Use the function ProMechsectiondataHollowcirclesectdataSet() to modify the hollow circle beam section data structure.
- PRO_MECH_BEAM_SECTION_SOLID_ELLIPSE—Specifies a solid ellipse beam section type. The cross section dimension for this beam type is specified by the length of the major axis and the length of the minor axis. Use the function ProMechsectiondataEllipsesectdataGet() to access the solid ellipse beam data structure. Use the function ProMechsectiondataEllipsesectdataSet() to modify the solid ellipse beam data structure.
- PRO_MECH_BEAM_SECTION_HOLLOW_ELLIPSE—Specifies a hollow ellipse beam section type. The cross section dimension for this beam type is specified by the length of the major axis, length of the minor axis, and the

inside major axis. Use the function `ProMechsectiondataHollowellipsesectdataGet()` to access the hollow ellipse beam data structure. Use the function `ProMechsectiondataHollowellipsesectdataSet()` to access the hollow ellipse beam data structure.

- `PRO_MECH_BEAM_SECTION_GENERAL`—Specifies a general beam section type. A general beam section type does not have a predefined shape. Use the function `ProMechsectiondataGeneralsectdataGet()` to access the general beam data structure. Use the function `ProMechsectiondataGeneralsectdataSet()` to access the general beam data structure.

The function `ProMechbeamsectionDescriptionGet()` returns the description for the specified beam section. Use the function `ProMechbeamsectionDescriptionSet()` to set the description of the specified beam section.

The function `ProMechsectiondataNonstructcogGet()` returns the Y and Z coordinates of the non-structural mass center of gravity. A non-structural mass is a mass that responds to gravity, but does not strengthen the structure. Non-structural masses can have different moments of inertia and gravitational centers than the specified beam. The function `ProMechsectiondataNonstructcogSet()` sets the Y and Z coordinates of the non-structural mass center of gravity.

The function `ProMechsectiondataNonstructmassGet()` returns the non-structural mass per unit length. Use the function `ProMechsectiondataNonstructmassSet()` to set the non-structural mass per unit length the beam section.

The function `ProMechsectiondataNonstructmomentGet()` returns the non-structural mass moment of inertia per unit length. Use the function `ProMechsectiondataNonstructmomentSet()` to set the non-structural mass moment of inertia per unit length of the beam section.

The function `ProMechsectiondataWarpcoeffGet()` returns the warp coefficient of the beam section. Use the function `ProMechsectiondataWarpcoeffSet()` to set the warp coefficient of the beam section.

The function `ProMechsectiondataFree()` releases the memory assigned to the beam section data.

Sketched Beam Section

The sketched beam section functions described below have been deprecated. Use the functions described in the section Creo Simulate Beams: Sections, Sketched Sections, and General Sections on page 1898 instead.

Functions Superseded:

- **ProMechsketchedsctndataAlloc()**
- **ProMechsketchedsctndataFeatureidGet()**
- **ProMechsketchedsctndataOrienttypeGet()**
- **ProMechsketchedsctndataOrienttypeSet()**
- **ProMechsketchedsctndataShearcenterGet()**
- **ProMechsketchedsctndataShearcenterSet()**
- **ProMechsketchedsctndataFree()**

The function `ProMechsketchedsctndataAlloc()` allocates the memory for the sketched beam section data handle.

The function `ProMechsketchedsctndataFeatureidGet()` returns the sketch feature id of the sketched beam section.

The function `ProMechsketchedsctndataOrienttypeGet()` returns the type of orientation for the sketched beam section. The beam orientation defines the Y direction of the beam, that is, how it rotates on the XY plane. The types of orientation is as follows:

- `PRO_MECH_BEAM_SECTION_SKET_XY_AS_YZ`—The beam section X and Y coordinates correspond to the beam Y and Z directions, respectively.
- `PRO_MECH_BEAM_SECTION_SKET_XY_AS_ZY`—The beam section X and Y coordinates correspond to the beam Z and Y directions, respectively.

Use the function `ProMechsketchedsctndataOrienttypeSet()` to set the type of orientation for the sketched beam section.

The function `ProMechsketchedsctndataShearcenterGet()` returns the shear center of the beam section. The shear center is the point on a beam section about which the section rotates under deflection. The function `ProMechsketchedsctndataShearcenterSet()` sets the shear center of the beam section.

The function `ProMechsketchedsctndataFree()` releases the memory assigned to sketched beam section data.

General Beam Section

The general beam section functions described below have been deprecated. Use the functions described in the section Creo Simulate Beams: Sections, Sketched Sections, and General Sections on page 1898 instead.

Functions Superseded:

- **ProMechgeneralsctndataAlloc()**
- **ProMechgeneralsctndataAreaGet()**
- **ProMechgeneralsctndataAreaSet()**
- **ProMechgeneralsctndataAreaproductGet()**
- **ProMechgeneralsctndataAreaproductSet()**
- **ProMechgeneralsctndataMomentsGet()**
- **ProMechgeneralsctndataMomentsSet()**
- **ProMechgeneralsctndataShearcenterGet()**
- **ProMechgeneralsctndataShearcenterSet()**
- **ProMechgeneralsctndataShearfactorGet()**
- **ProMechgeneralsctndataShearfactorSet()**
- **ProMechgeneralsctndataTorsionstiffnessGet()**
- **ProMechgeneralsctndataTorsionstiffnessSet()**
- **ProMechgeneralsctndataStressrecoverypntsGet()**
- **ProMechgeneralsctndataStressrecoverypntsSet()**
- **ProMechgeneralsctndataFree()**

The function `ProMechgeneralsctndataAlloc()` allocates the memory for the general beam section data handle.

The function `ProMechgeneralsctndataAreaGet()` returns the cross-sectional area for each beam section. Use the function `ProMechgeneralsctndataAreaSet()` to set the cross-sectional area for the beam section.

The function `ProMechgeneralsctndataAreaproductGet()` returns the area product of the moments of inertia. Use the function `ProMechgeneralsctndataAreaproductSet()` to set the area product of inertia.

The function `ProMechgeneralsctndataMomentsGet()` returns the second moments of area for each beam section. These properties describe the stiffness in bending about a beam's principle Y and Z axes. Use the function `ProMechgeneralsctndataMomentsSet()` to set the second moments of area for each beam section.

The function `ProMechgeneralsctndataShearcenterGet()` returns the Shear DY and Shear DZ values. These values specify the distance between shear center (the point on a beam section about which the section rotates under deflection) and the centroid of the beam section, with respect to the principal axes. Use the function `ProMechgeneralsctndataShearcenterSet()` to set the Shear DY and Shear DZ values.

The function `ProMechgeneralsctndataShearfactorGet()` returns the Shear FY and Shear FZ values for the beam section. These values represent the ratio of a beam's effective "shear area" to its true cross-sectional area for shear in the principal Y and Z directions. Use the function `ProMechgeneralsctndataShearfactorSet()` to set the Shear FY and Shear FZ values for the beam section.

The function `ProMechgeneralsctndataTorsionstiffnessGet()` returns the second polar moment of area for each beam section. This property describes the stiffness in torsion. Use the function `ProMechgeneralsctndataTorsionstiffnessSet()` to set the torsion stiffness for each beam section.

The function `ProMechgeneralsctndataStressrecoverypntsGet()` returns the stress recovery points of the beam section. Use the `ProMechgeneralsctndataStressrecoverypntsSet()` to set the stress recovery points of the beam section.

The function `ProMechgeneralsctndataFree()` releases the memory assigned to the sketched beam section data.

Beam Orientations

The functions described in this section provide read and write access to the data and contents of the Creo Simulate beam orientations data structure. Beam orientations use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_BEAM_ORIENT`.

Functions Introduced:

- **ProMechbeamorientDescriptionGet()**
- **ProMechbeamorientDescriptionSet()**
- **ProMechBeamorientExpressionGet()**
- **ProMechBeamorientExpressionSet()**
- **ProMechBeamorientIntegerGet()**
- **ProMechBeamorientIntegerSet()**
- **ProMechBeamorientVectorGet()**
- **ProMechBeamorientVectorSet()**

Functions Superseded:

- **ProMechbeamorientDataGet()**
- **ProMechbeamorientDataSet()**
- **ProMechbeamorientdataTypeGet()**
- **ProMechbeamorientdataTypeSet()**

- **ProMechbeamorientdataAngleGet()**
- **ProMechbeamorientdataAngleSet()**
- **ProMechbeamorientdataVectorGet()**
- **ProMechbeamorientdataVectorSet()**
- **ProMechbeamorientdataFree()**

The function `ProMechbeamorientDataGet()` provides access to the data structure containing the properties of the beam orientation. Use the function `ProMechbeamorientDataSet()` to set the properties of the beam orientation data. The functions `ProMechbeamorientDataGet()` and `ProMechbeamorientDataSet()` have been deprecated. Use the functions `ProMechBeamorientIntegerGet()` and `ProMechBeamorientIntegerSet()` instead.

The function `ProMechbeamorientDescriptionGet()` returns the description of the beam orientation. Use the function `ProMechbeamorientDescriptionSet()` to set the description of the beam orientation.

The functions `ProMechBeamorientExpressionGet/Set`, `ProMechBeamorientIntegerGet/Set`, and `ProMechBeamorientVectorGet/Set` use the enumerated type `ProMechBeamorientPropertyType` to define the beam orientation property type.

The values are as follows:

```
typedef enum
{
  PRO_MECH_BEAM_ORIENT_OFFSET_TYPE = 0,  /* ProMechBeamOrientType (int) */

  PRO_MECH_BEAM_ORIENT_ANGLE       = 1,  /* ProMechExpression */

  PRO_MECH_BEAM_ORIENT_OFFSET      = 2   /* ProMechVector (3)
                                            [ Dx Dy Dz ]
                                        */
}
ProMechBeamorientPropertyType;
```

The function `ProMechBeamorientExpressionGet()` gets the value for the defined beam orientation property type. If an expression is defined, the output value is calculated using `ProMathExpressionEvaluate()`. Use the function `ProMechBeamorientExpressionSet()` to set the value for the beam orientation property type. For the functions `ProMechBeamorientExpressionGet/Set`, the only enumerated value allowed is `PRO_MECH_BEAM_ORIENT_ANGLE`. You can get and set the angle of the beam orientation.

The function ProMechBeamorientIntegerGet() gets the integer of the specified beam orientation property type. Use the function ProMechBeamorientIntegerSet() to set an integer value for the specified beam orientation property type. For the functions `ProMechBeamorientIntegerGet/Set`, the only enumerated value allowed is `PRO_MECH_BEAM_ORIENT_OFFSET_TYPE`.

`PRO_MECH_BEAM_ORIENT_OFFSET_TYPE` returns the handle to `ProMechBeamOrientType`. You can specify the type of orientation using the enumerated type `ProMechBeamOrientType`.

The types of orientations are as follows:

- `PRO_MECH_BEAM_ORIENT_OFFSET_SHAPE_ORIGIN`— Specifies the point of origin of the beam shape coordinate system.
- `PRO_MECH_BEAM_ORIENT_OFFSET_CENTROID`— Specifies the origin of the principal coordinate system which is at the centroid of the section. For general sections and all standard sections, it is coincident with `PRO_MECH_BEAM_ORIENT_OFFSET_SHAPE_ORIGIN`.
- `PRO_MECH_BEAM_ORIENT_OFFSET_SHEAR_CENTER`— Specifies the point on a beam section about which the section rotates under deflection.

The function `ProMechBeamorientVectorGet()` returns the Creo Simulate vector handle `ProMechVector`. Use the function `ProMechBeamorientVectorSet()` to set the handle to `ProMechVector`. The functions `ProMechVectorComponentGet/Set` provide read and write access to the individual vector components of the specified beam orientation property type. For more information about handling vector components, see Creo Simulate Vector Functions on page 1895.

For the functions `ProMechBeamorientVectorGet/Set`, the only enumerated value allowed is `PRO_MECH_BEAM_ORIENT_OFFSET`. For example, you can get/set the direction of the beam vector.

The function `ProMechBeamorientVectorGet()` automatically allocates the memory for the Creo Simulate vector handle. Use the function `ProMechVectorFree()` to free the memory.

The function `ProMechbeamorientdataTypeGet()` returns the type of the orientation specified for the Creo Simulate beam orientation item. Use the function `ProMechbeamorientdataTypeSet()` to set the type of orientation The functions `ProMechbeamorientdataTypeGet()` and `ProMechbeamorientdataTypeSet()` have been deprecated. Use the enumerated type `ProMechBeamOrientType` instead.

The function `ProMechbeamorientdataAngleGet()` returns the angle of the beam orientation. Use the function `ProMechbeamorientdataAngleSet()` to set the angle of the beam orientation. The functions `ProMechbeamorientdataAngleGet()` and

ProMechbeamorientdataAngleSet() have been deprecated. Use the functions ProMechBeamorientExpressionGet() and ProMechBeamorientExpressionSet() instead.

The function ProMechbeamorientdataVectorGet() returns the direction of the beam vector. Use the function ProMechbeamorientdataVectorSet() to set the direction of the beam vector. The functions ProMechbeamorientdataVectorGet() and ProMechbeamorientdataVectorSet() have been deprecated. Use the functions ProMechBeamorientVectorGet() and ProMechBeamorientVectorSet() instead.

Use the function ProMechbeamorientdataFree() to free the memory allocated to the beam orientation data structure. The function ProMechbeamorientdataFree() has been deprecated.

Beam Releases

The beam releases specify the degrees of freedom you want to release for a beam end or beam ends. Beam releases determine the degrees of freedom that do not participate in a connection at the end of a beam. You can specify beam releases for both straight and curved beams.

The beam releases data structure contains a combination of the six degrees of freedom relative to the beam's local axes, that is, translation in X, Y, and Z and rotation in X, Y, and Z.

The functions described in this section provide read and write access to the data and contents of the Creo Simulate beam releases data structure. Beam releases use the ProType field in the ProMechitem structure as PRO_SIMULATION_BEAM_RELEASE.

Functions Introduced:

- **ProMechbeamreleasedataAlloc()**
- **ProMechbeamreleaseDataGet()**
- **ProMechbeamreleaseDataSet()**
- **ProMechbeamreleaseDescriptionGet()**
- **ProMechbeamreleaseDescriptionSet()**
- **ProMechbeamreleasedataRotationflagsGet()**
- **ProMechbeamreleasedataRotationflagsSet()**
- **ProMechbeamreleasedataTranslationflagsGet()**
- **ProMechbeamreleasedataTranslationflagsSet()**
- **ProMechbeamreleasedataFree()**

The function `ProMechbeamreleasedataAlloc()` allocates the memory for the beam release data handle.

The function `ProMechbeamreleaseDataGet()` provides access to the data structure containing the properties of the beam release item.

The translation and rotation flags correspond to translation or rotation about the X, Y, and Z directions. Set the value of each flag to `PRO_B_TRUE`, to indicate that the beam is not constrained in the specified direction. Set the value to `PRO_B_FALSE` to indicate that the beam is constrained in the specified direction.

The function `ProMechbeamreleasedataFree()` releases the memory assigned to the beam release data handle.

Creo Simulate Spring Items

A spring connects two points or a point to the ground in the specified model. It provides the stiffness that you specify at the location on the model where you place it. The stiffness can be translational (force per unit length) or torsional (torque). The force generated by the spring is proportional to the amount of displacement that occurs. The functions described in this section enable you to access the data and contents of the Creo Simulate spring items. Springs use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_SPRING`.

Functions Introduced:

- **ProMechspringReferencesGet()**
- **ProMechspringReferencesSet()**
- **ProMechspringTypeGet()**
- **ProMechspringAdvanceddataGet()**
- **ProMechspringAdvanceddataSet()**
- **ProMechspringGrounddataGet()**
- **ProMechspringGrounddataSet()**
- **ProMechspringSimpledataGet()**
- **ProMechspringSimpledataSet()**
- **ProMechsimplespringdataAlloc()**
- **ProMechsimplespringdataTorsionalstiffnessGet()**
- **ProMechsimplespringdataTorsionalstiffnessSet()**
- **ProMechsimplespringdataExtensionalstiffnessValueGet()**
- **ProMechsimplespringdataExtensionalstiffnessValueSet()**
- **ProMechsimplespringdataFree()**

- **ProMechadvancedspringdataAlloc()**
- **ProMechadvancedspringdataPropertiesGet()**
- **ProMechadvancedspringdataPropertiesSet()**
- **ProMechadvancedspringdataRotationGet()**
- **ProMechadvancedspringdataRotationSet()**
- **ProMechadvancedspringdataYdirectionGet()**
- **ProMechadvancedspringdataYdirectionSet()**
- **ProMechadvancedspringdataFree()**
- **ProMechgroundspringdataAlloc()**
- **ProMechgroundspringdataCsysGet()**
- **ProMechgroundspringdataCsysSet()**
- **ProMechgroundspringdataPropertiesGet()**
- **ProMechgroundspringdataPropertiesSet()**
- **ProMechgroundspringdataFree()**

The function `ProMechspringReferencesGet()` returns the geometrical references specified while modeling the spring. The references define the location of the spring on the model. Use the function `ProMechspringReferencesSet()` to set the geometrical references for the spring.

The function `ProMechspringTypeGet()` returns the type of the specified spring. The output argument *Type* has one of the following values:

- `PRO_MECH_SPRING_SIMPLE`—Specifies a simple spring. This type of spring connects two points, two vertices, a point to an edge, a point to a surface, a point to a pattern of points, a point to a single point feature. The extensional and torsional stiffness properties will be defined for this spring.

  Use the function `ProMechspringSimpledataGet()` to provide access to the data structure containing the simple spring data. Use the function `ProMechspringSimpledataSet()` to modify the data structure containing the simple spring data.
- `PRO_MECH_SPRING_GROUND`—Specifies a To Ground spring. This type of string connects a point, a single point feature, or a single pattern of points to ground. The spring stiffness properties and the orientation are defined for this spring.

  A separate properties object can be defined for this type of spring.

  Use the function `ProMechspringGrounddataGet()` to provide access to the data structure containing the ground spring data. Use the function

ProMechspringGrounddataSet() to modify the data structure containing the ground spring data.

- PRO_MECH_SPRING_ADVANCED—Specifies an Advanced spring. This type of spring connects two points, a point to an edge, a point to a surface, a point to a pattern of points, or a point to a single point feature. The stiffness properties, orientation properties, and an additional rotation are defined for this spring. Use the function ProMechspringAdvanceddataGet() to provide access to the data structure containing the advanced spring data. Use the function ProMechspringAdvanceddataSet() to modify the data structure containing the advanced spring data.

Extensional stiffness of a spring resists the stretching or compression of the spring. The extensional stiffness of the spring is of constant stiffness or is defined by a force-deflection curve. The function ProMechsimplespringdataExtensionalstiffnessValueGet() returns the extensional stiffness of the spring. Use the function ProMechsimplespringdataExtensionalstiffnessValueSet() to set the extensional stiffness for the spring.

Note

Note: The functions ProMechsimplespringdataExtensionalstiffnessGet() and ProMechsimplespringdataExtensionalstiffnessSet() have been deprecated. Use the functions ProMechsimplespringdataExtensionalstiffnessValue Get() and ProMechsimplespringdataExtensionalstiffnessValue Set() instead.

Creo Simulate Spring Property Items

The functions described in this section provide access to the data and contents of the Creo Simulate spring property items. Spring properties use the ProType field in the ProMechitem structure as PRO_SIMULATION_SPRING_PROPS.

Functions Introduced:

- **ProMechspringpropsDescriptionGet()**
- **ProMechspringpropsDescriptionSet()**
- **ProMechSpringpropsBooleanGet()**
- **ProMechSpringpropsBooleanSet()**

- **ProMechSpringpropsMatrixGet()**
- **ProMechSpringpropsMatrixSet()**
- **ProMechSpringpropsVectorGet()**
- **ProMechSpringpropsVectorSet()**

Functions Superseded:

- **ProMechspringpropsdataAlloc()**
- **ProMechspringpropsDataGet()**
- **ProMechspringpropsDataSet()**
- **ProMechspringpropsdataDampingcoefficientsGet()**
- **ProMechspringpropsdataDampingcoefficientsSet()**
- **ProMechspringpropsdataExtensionalcoefficientsGet()**
- **ProMechspringpropsdataExtensionalcoefficientsSet()**
- **ProMechspringpropsdataTorsionalcoefficientsGet()**
- **ProMechspringpropsdataTorsionalcoefficientsSet()**
- **ProMechspringpropsdataCouplingcoefficientsGet()**
- **ProMechspringpropsdataCouplingcoefficientsSet()**
- **ProMechspringpropsdataAutocouplingGet()**
- **ProMechspringpropsdataAutocouplingSet()**
- **ProMechspringpropsdataFree()**

The function `ProMechspringpropsDataGet()` provides access to the data structure containing the Spring Properties data. The function `ProMechspringpropsDataGet()` has been deprecated. Use either `ProMechspringpropsMatrixGet()` or `ProMechspringpropsVectorGet()` instead.

The functions listed above provide read and write access to the definition of the spring properties. You can access the name, description, extensional, torsional and coupling stiffness, and the damping coefficients for the spring properties.

The function `ProMechspringpropsDescriptionGet()` returns the description for the spring property. Use the function `ProMechspringpropsDescriptionSet()` to set the description for the spring property.

The functions `ProMechSpringpropsBooleanGet/Set`, `ProMechSpringpropsMatrixGet/Set`, and `ProMechSpringpropsVectorGet/Set` use the enumerated type `ProMechSpringpropsPropertyType` to define the spring property type.

The enumerated type `ProMechSpringpropsPropertyType` has the following values:

- `PRO_MECH_SPRINGPROPS_EXTENSIONAL`
- `PRO_MECH_SPRINGPROPS_TORSIONAL`
- `PRO_MECH_SPRINGPROPS_COUPLING`
- `PRO_MECH_SPRINGPROPS_DAMPING`
- `PRO_MECH_SPRINGPROPS_AUTOCOUPLING`

The function `ProMechSpringpropsBooleanGet()` gets the Boolean value for the autocoupling option. Use the function `ProMechSpringpropsBooleanSet()` to set the Boolean value for the autocoupling option.

The function `ProMechSpringpropsMatrixGet()` returns the Creo Simulate matrix handle `ProMechMatrix`. Use the function `ProMechSpringpropsMatrixSet()` to set the handle to `ProMechMatrix`. The functions `ProMechMatrixComponentGet/Set` provide read and write access to the individual matrix components. For more information about handling matrix components, see Creo Simulate Matrix Functions on page 1894.

The output matrix gets the following values depending on the spring property type:

- Extensional coefficients (Kxx,Kyy,Kzz,Kxy,Kxz,Kyz)
- Torsional coefficients (Txx,Tyy,Tzz,Txy,Txz,Tyz)
- Coupling coefficients (KTxx, KTxy, KTxz, KTyx, KTyy, KTyz, KTzx, KTzy, KTzz)

The function `ProMechSpringpropsMatrixGet()` automatically allocates memory for the Creo Simulate matrix handle. Use the function `ProMechMatrixFree()` to free the assigned memory.

The function `ProMechSpringpropsVectorGet()` returns the Creo Simulate vector handle `ProMechVector`. Use the function `ProMechSpringpropsVectorSet()` to set the handle to `ProMechVector`. The functions `ProMechVectorComponentGet/Set` provide read and write access to the individual vector components. For more information about handling vector components, see Creo Simulate Vector Functions on page 1895.. The vector returns the damping coefficients (Cxx, Cyy, Czz) as output. The function `ProMechSpringpropsVectorGet()` automatically allocates the memory for the Creo Simulate vector handle. Use the function `ProMechVectorFree()` to free the memory.

Creo Simulate Mass Items

A mass is an idealization that you can use to represent a concentrated mass without a specified shape. The mass of an object determines how that object resists translation and rotation. You can also add mass that is distributed over features such as curves, edges, or surfaces.

The functions described in this section provide access to the data and contents of the Creo Simulate mass items. Masses use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_MASS`.

Functions Introduced:

- **ProMechmassDistributionGet()**
- **ProMechmassDistributionSet()**
- **ProMechmassReferencesGet()**
- **ProMechmassReferencesSet()**
- **ProMechmassTypeGet()**
- **ProMechmassSimpledataGet()**
- **ProMechmassSimpledataSet()**
- **ProMechmassAdvanceddataGet()**
- **ProMechmassAdvanceddataSet()**
- **ProMechmassComponentdataGet()**
- **ProMechmassComponentpointdataSet()**
- **ProMechmassComponentdistributeddataSet()**
- **ProMechsimplemassdataAlloc()**
- **ProMechsimplemassdataMassGet()**
- **ProMechsimplemassdataMassSet()**
- **ProMechsimplemassdataFree()**
- **ProMechadvancedmassdataAlloc()**
- **ProMechadvancedmassdataCsysGet()**
- **ProMechadvancedmassdataCsysSet()**
- **ProMechadvancedmassdataPropertiesGet()**
- **ProMechadvancedmassdataPropertiesSet()**
- **ProMechadvancedmassdataFree()**
- **ProMechcomponentmassdataAlloc()**
- **ProMechcomponentmassdataComponentGet()**

- **ProMechcomponentmassdataComponentSet()**
- **ProMechcomponentmassdataFree()**

The function `ProMechmassDistributionGet()` returns the types of masses that can be applied to curves, surfaces, and edges in FEM Mode.

The output argument *type* can have the following values:

- `PRO_MECH_MASS_DISTR_AT_POINT`—Specifies that the mass is added to a point, vertex, multiple single points, point features, and point patterns.
- `PRO_MECH_MASS_DISTR_TOTAL`—Specifies the total distribution of mass along a surface or curve.
- `PRO_MECH_MASS_DISTR_PER_UNIT`—Specifies the distribution of mass along a curve or surface per unit length or per unit area respectively.

Use the function `ProMechmassDistributionSet()` to set the types of masses that can be applied to curves, surfaces, and edges in FEM Mode.

The function `ProMechmassReferencesGet()` returns the mass reference objects. The references can be either curves, edges, or surfaces. Use the function `ProMechmassReferencesSet()` to set the mass reference objects.

The function `ProMechmassTypeGet()` returns the types of masses defined for a point or vertex. The output argument *type* can have one of the following values:

- `PRO_MECH_MASS_SIMPLE`—Specifies a simple mass type. Specify an integer as the mass value and points, single point, multiple single points, point features, point patterns, edges, curves, or surfaces as the reference for the simple mass type. Use the function `ProMechmassSimpledataGet()` to access the simple mass data structure. Use the function `ProMechmassSimpledataSet()` to modify the simple mass data structure.
- `PRO_MECH_MASS_ADVANCED`—Specifies the advanced mass type. Specify the coordinate system and the mass property object for a single point, multiple single points, point features, and point patterns. Use the function `ProMechmassAdvanceddataGet()` to access the advanced mass data structure. Use the function `ProMechmassAdvanceddataSet()` to modify the advanced mass data structure.
- `PRO_MECH_MASS_COMP_AT_POINT`—Specifies the component mass data for a part or subassembly of an assembly. For this type of mass, the mass definition is specified using the component's mass, moment of inertia, and center of gravity. This mass type can be created using points, edges or curves, or Surfaces as the reference. Use the function `ProMechmassComponentdataGet()` to access the component mass data structure. Use the function `ProMechmassComponentpointdataSet()` to modify the component at point mass data structure.

- This mass type is applicable only for the assembly mode.
- PRO_MECH_MASS_COMP_DISTRIBUTED—Specifies the component distributed mass data for a part or subassembly of an assembly. For this type of mass, only the component's mass is used to specify the mass definition. This mass type can be created using points, edges or curves, or surfaces as the reference. Use the function ProMechmassComponentdataGet() to access the component mass data structure. Use the function ProMechmassComponentdistributeddataSet() to modify the component distributed mass data structure.
- This mass type is applicable only for the assembly mode.

The function ProMechsimplemassdataAlloc() allocates memory for a simple mass data structure.

The function ProMechsimplemassdataMassGet() returns the value of the mass for a simple mass data and the function ProMechsimplemassdataMassSet() sets the value of the mass for a simple mass data.

The function ProMechadvancedmassdataAlloc() allocates memory for the advanced mass data structure.

The function ProMechadvancedmassdataCsysGet() returns the reference co-ordinate system for the advanced mass data and the function ProMechadvancedmassdataCsysSet() sets the reference co-ordinate system for the advanced mass data.

The function ProMechadvancedmassdataPropertiesGet() returns the mass property for the advanced mass data.

The function ProMechadvancedmassdataPropertiesSet() sets the mass property for the advanced mass data.

The function ProMechcomponentmassdataAlloc() allocates memory for the component mass data structure.

The function ProMechcomponentmassdataComponentGet() specifies a reference for the component mass data. You can specify only one datum point or vertex as reference for the mass type PRO_MECH_MASS_COMP_AT_POINT. You can specify points, edges, curves, or surfaces as reference for the mass type PRO_MECH_MASS_COMP_DISTRIBUTED.

The function ProMechcomponentmassdataComponentSet() sets the reference type for the component mass data.

Creo Simulate Mass Properties

The functions described in this section provide read and write access to the data and contents of the Creo Simulate property items. Mass properties use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_MASS_PROPS`.

Functions Introduced:

- **ProMechmasspropsDescriptionGet()**
- **ProMechmasspropsDescriptionSet()**
- **ProMechMasspropsExpressionGet()**
- **ProMechMasspropsExpressionSet()**
- **ProMechMasspropsMatrixGet()**
- **ProMechMasspropsMatrixSet()**

Functions Superseded:

- **ProMechmasspropsMassGet()**
- **ProMechmasspropsMassSet()**
- **ProMechmasspropsMomentsGet()**
- **ProMechmasspropsMomentsSet()**

The function `ProMechmasspropsDescriptionGet()` returns the description for the mass property. Use the function `ProMechmasspropsmassSet()` to set the description for the mass property.

The function `ProMechmasspropsMassGet()` returns the value of the mass specified for the mass properties object. Use the function `ProMechmasspropsMassSet()` to set the value of the mass. The functions `ProMechmasspropsMassGet()` and `ProMechmasspropsMassSet()` have been deprecated. Use the functions `ProMechMasspropsExpressionGet()` and `ProMechMasspropsExpressionSet()` instead.

The function `ProMechmasspropsMomentsGet()` returns the moment of inertia about each mass element's center of gravity with respect to the axes and principal planes of the WCS. The moment of inertia is returned in the form of a matrix. Use the function `ProMechmasspropsMomentsSet()` to set the moment of inertia matrix. The functions `ProMechmasspropsMomentsGet()` and `ProMechmasspropsMomentsSet()` have been deprecated. Use the functions `ProMechMasspropsMatrixGet()` and `ProMechMasspropsMatrixSet()` instead

The functions ProMechMasspropsExpressionGet() and ProMechMasspropsExpressionSet() use the enumerated type ProMechMasspropsPropertyType. The values are:

- PRO_MECH_MASSPROPS_MASS
- PRO_MECH_MASSPROPS_MOMENTS

The function ProMechMasspropsExpressionGet() gets the value of the defined mass property object. If an expression is defined, the output value can be calculated using ProMathExpressionEvaluate(). Use the function ProMechMasspropsExpressionSet() to set the value for the mass property object.

The function ProMechMasspropsMatrixGet() returns the Creo Simulate matrix handle ProMechMatrix. Use the function ProMechMasspropsMatrixSet() to set the handle to ProMechMatrix. The functions ProMechMatrixComponentGet/Set provide read and write access to the individual matrix components. For more information about handling matrix components, see Creo Simulate Matrix Functions on page 1894.

Creo Simulate Material Assignment

The functions described in this section allow you to assign materials to the 2D and 3D models. Material assignment uses the ProType field in the ProMechitem structure as PRO_SIMULATION_MATL_ASSIGN.

Functions referring to the material assignment elements use the structure ProMechMatlAssignData which is defined as:

```
typedef struct pro_matlassign_data
 {
   int  matl_id;
   int  matl_orient_id;
  }
  ProMatlassignData;
```

Accessing ProMechmatlassign

Functions Introduced:

- **ProMechmatlassignReferencesGet()**
- **ProMechmatlassignReferencesSet()**
- **ProMechmatlassignDataGet()**
- **ProMechmatlassignDataSet()**

The function `ProMechmatlassignReferencesGet()` returns the model references for material assignment. Use the function `ProMechmatlassignReferencesSet()` to set the model references for material assignment.

The function `ProMechmatlassignDataGet()` provides access to the data structure containing the properties of the material assignment item. Use the function `ProMechmatlassignDataSet()` to set the data structure containing the properties of the material assignment item.

Material Assignment Data

Functions Introduced:

- **ProMechmecttempdataAlloc()**
- **ProMechmatlassigndataFree()**
- **ProMechmatlassigndataMaterialidGet()**
- **ProMechmatlassigndataMaterialidSet()**
- **ProMechmatlassigndataMaterialorientidGet()**
- **ProMechmatlassigndataMaterialorientidSet()**

The function `ProMechmecttempdataAlloc()` allocates the memory for a Creo Simulate material assignment data handle.

The function `ProMechmatlassigndataFree()` releases the memory assigned to the Creo Simulate material assignment data handle.

The function `ProMechmatlassigndataMaterialidGet()` returns the material id defined for the model references. Use the function `ProMechmatlassigndataMaterialidSet()` to set the material id for the model references.

The function `ProMechmatlassigndataMaterialorientidGet()` returns the material orientation id of the model references. Use the function `ProMechmatlassigndataMaterialorientidSet()` to set the material orientation id for the model references.

Material Orientations

The functions described in this section specify material orientation for surfaces, volumes, shells, solids, 2D solids, and 2D plates. These functions provide read and write access to the data and contents of Creo Simulate material orientation objects. Material orientations use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_MATL_ORIENT`.

Functions Introduced:

- **ProMechmaterialorientdataAlloc()**
- **ProMechmaterialorientDataGet()**
- **ProMechmaterialorientDataSet()**
- **ProMechmaterialorientDescriptionGet()**
- **ProMechmaterialorientDescriptionSet()**
- **ProMechmaterialorientdataObjecttypeGet()**
- **ProMechmaterialorientdataObjecttypeSet()**
- **ProMechmaterialorientdataCsysdataGet()**
- **ProMechmaterialorientdataCsysdataSet()**
- **ProMechmaterialorientdataProjectiondataGet()**
- **ProMechmaterialorientdataProjectiondataSet()**
- **ProMechmaterialorientdataRotationGet()**
- **ProMechmaterialorientdataRotationSet()**
- **ProMechmaterialorientdataSurfacerotationGet()**
- **ProMechmaterialorientdataSurfacerotationSet()**
- **ProMechmaterialorientdataTypeGet()**
- **ProMechmaterialorientdataSurfacerotationUnset()**
- **ProMechmaterialorientdataFirstdirectionSet()**
- **ProMechmaterialorientdataSeconddirectionSet()**
- **ProMechmaterialorientdataFree()**
- **ProMechmaterialorientcsysAlloc()**
- **ProMechmaterialorientcsysCsysGet()**
- **ProMechmaterialorientcsysCsysSet()**
- **ProMechmaterialorientcsysProjectiontypeGet()**
- **ProMechmaterialorientcsysProjectiontypeSet()**
- **ProMechmaterialorientcsysXaxisGet()**
- **ProMechmaterialorientcsysXaxisSet()**
- **ProMechmaterialorientcsysYaxisGet()**
- **ProMechmaterialorientcsysYaxisSet()**
- **ProMechmaterialorientcsysZaxisGet()**
- **ProMechmaterialorientcsysZaxisSet()**
- **ProMechmaterialorientcsysFree()**
- **ProMechmaterialorientprojAlloc()**

- **ProMechmaterialorientprojPointsGet()**
- **ProMechmaterialorientprojPointsSet()**
- **ProMechmaterialorientprojTypeGet()**
- **ProMechmaterialorientprojXyzvectorGet()**
- **ProMechmaterialorientprojXyzvectorSet()**
- **ProMechmaterialorientprojFree()**

The function `ProMechMaterialorientDataGet()` provides access to the data structure containing the properties of the material orientation item. Use the function `ProMechmaterialorientDataSet()` to set the data structure containing the properties of the material orientation item.

The function `ProMechmaterialorientDescriptionGet()` returns the description of the material orientation. Use the function `ProMechmaterialorientDescriptionSet()` to set the description of the material orientation.

The function `ProMechmaterialorientdataObjecttypeGet()` returns the type of object to which the material orientation is applied. The types of object are:

- `PRO_MECH_MATLORI_MODEL`—Specifies a model.
- `PRO_MECH_MATLORI_SURFACE`—Specifies a surface.

Use the function `ProMechmaterialorientdataObjecttypeSet()` to set the type of object.

The function `ProMechmaterialorientReferencesGet()` returns the geometric references specified for the material orientation object.

The function `ProMechmaterialorientdataCsysdataGet()` returns a handle to the data structure containing the coordinate system data for the material orientation. Use the function `ProMechmaterialorientdataCsysdataSet()` to set the coordinate system data for the material orientation.

The function `ProMechmaterialorientdataTypeGet()` returns the type of the material direction. The types are as follows:

- `PRO_MECH_MATLORI_COORD_SYSTEM`—Specifies that the material orientation direction is determined by the reference coordinate system.
- `PRO_MECH_MATLORI_1_DIR`—Specifies that the material orientation direction is determined by the first parametric direction of the material. Use the function `ProMechmaterialorientdataFirstdirectionSet()` to set the first direction of the material as the orientation type.
- `PRO_MECH_MATLORI_2_DIR`—Specifies that the material orientation direction is determined by the second parametric direction of the material. Use

the function ProMechmaterialorientdataSeconddirectionSet() to set the second direction of the material as the orientation type.

- PRO_MECH_MATLORI_PROJ_VECTOR—Specifies that the material orientation direction is determined by the projection vector.

The function ProMechmaterialorientdataProjectiondataGet() returns the structure containing the projection data for the material orientation. Use the function ProMechmaterialorientdataProjectiondataSet() to set the projection data for the material orientation.

The function ProMechmaterialorientdataRotationGet() returns additional rotations about one or more material directions only if the material orientation type is PRO_MECH_MATLORI_MODEL. Use the function ProMechmaterialorientdataRotationSet() to set the additional rotations about one or more material directions only if the material orientation type is PRO_MECH_MATLORI_MODEL.

The function ProMechmaterialorientdataSurfacerotationGet() returns the rotation angle for the material orientation if the orientation type is PRO_MECH_MATLORI_SURFACE.

Use the function ProMechmaterialorientdataSurfacerotationSet() to set the rotation angle for the material orientation if the orientation type is PRO_MECH_MATLORI_SURFACE.

Use the function ProMechmaterialorientdataSurfacerotationUnset() removes the rotation angle for the material orientation if the orientation type is PRO_MECH_MATLORI_SURFACE.

The function ProMechmaterialorientprojTypeGet() returns the type of projection assigned to the material orientation. The types of materials are:

- PRO_MECH_MATLORI_PROJ_XYZ—Specifies the values for the X, Y, and Z components to define the projection vector for the material orientation. Use the function ProMechmaterialorientprojXyzvectorGet() to access the projection vector. Use the function ProMechmaterialorientprojXyzvectorSet() to set the projection vector for the material orientation.
- PRO_MECH_MATLORI_PROJ_POINTS—Specifies the two points used to define the projection vector for the material orientation. Use the function ProMechmaterialorientprojPointsGet() to access the two points. Use the function ProMechmaterialorientprojPointsSet() to set the two points used for projection.

The function ProMechmaterialorientcsysCsysGet() returns the coordinate system used to specify the material directions. Use the function ProMechmaterialorientcsysCsysSet() to set the coordinate system for the material directions.

The function ProMechmaterialorientcsysProjectiontypeGet() returns the projection vector for the material orientation object. The valid projection types are as follows:

- PRO_MECH_MATLORI_CSYS_PROJ_CLOSEST—Specifies the material Direction 1 through a series of calculations.
- PRO_MECH_MATLORI_CSYS_PROJ_X—Specifies that the material direction 1 is along the direction of the X axis of the referenced coordinate system projected onto the surface.

Use the function ProMechmaterialorientcsysProjectiontypeSet() to set the projection type for the material orientation data.

The function ProMechmaterialorientcsysXaxisGet() returns the material direction to which the x-axis of the coordinate system is mapped. Use the function ProMechmaterialorientcsysXaxisSet() to set the material direction to which the x-axis of the coordinate system is mapped.

The function ProMechmaterialorientcsysYaxisGet() returns the material direction to which the y-axis of the coordinate system is mapped. Use the function ProMechmaterialorientcsysYaxisSet() to set the material direction to which the y-axis of the coordinate system is mapped.

The function ProMechmaterialorientcsysZaxisGet() returns the material direction to which the z-axis of the coordinate system is mapped. Use the function ProMechmaterialorientcsysZaxisSet() to set the material direction to which the z-axis of the coordinate system is mapped.

Example 4: Creating Material Orientations Referencing a Selected Coordinate System

The sample code in the file PTMechExMatOrient.c located at <creo_toolkit_loadpoint>/protk_appls/pt_mech_examples/pt_mech_ex_src shows how to create a new material orientation item in the current model, referencing a selected coordinate system.

Creo Simulate Shells

Shells are used to model a thin layer of a defined thickness for a specified part. If the part is relatively thin compared to its length and width, use of shell modeling is more efficient.

The functions described in this section provide read and write access to the data and contents of the Creo Simulate shell objects. Shells use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_SHELL`.

Functions Introduced:

- **ProMechshellTypeGet()**
- **ProMechshellMaterialGet()**
- **ProMechshellMaterialIdGet()**
- **ProMechshellMaterialIdSet()**
- **ProMechshellReferencesGet()**
- **ProMechshellReferencesSet()**
- **ProMechshellSimpledataGet()**
- **ProMechshellSimpledataSet()**
- **ProMechshellAdvanceddataGet()**
- **ProMechshellAdvanceddataSet()**
- **ProMechshellsimpleAlloc()**
- **ProMechshellsimpleThicknessGet()**
- **ProMechshellsimpleThicknessSet()**
- **ProMechshellsimpleFree()**
- **ProMechshelladvancedAlloc()**
- **ProMechshelladvancedMaterialorientGet()**
- **ProMechshelladvancedMaterialorientSet()**
- **ProMechshelladvancedShellpropsGet()**
- **ProMechshelladvancedShellpropsSet()**
- **ProMechshelladvancedFree()**

The function `ProMechshellTypeGet()` returns the shell types. You can define the following types of shells:

- `PRO_MECH_SHELL_SIMPLE`—Specifies a simple shell of uniform thickness. Use the function `ProMechshellSimpledataGet()` to access the data structure for the simple shell. Use the function `ProMechshellSimpledataSet()` to modify the data structure for the simple shell.
- `PRO_MECH_SHELL_ADVANCED`—Specifies an advanced shell that uses specified shell properties. Use the function `ProMechshellAdvanceddataGet()` to access the data structure for the

advanced shells. Use the function `ProMechshellAdvanceddataSet()` to modify the data structure for the advanced shell.

The function `ProMechshellMaterialGet()` returns the material defined for the shell.

Use the function `ProMechshellMaterialIdSet()` to set the material id for the shell.

The function `ProMechshellReferencesGet()` returns the surfaces associated with the shell. Use the function `ProMechshellReferencesSet()` to set the references for the shell.

The function `ProMechshellsimpleThicknessGet()` returns the value of the thickness for the shell. Use the function `ProMechshellsimpleThicknessSet()` to set value of the thickness of the shell.

The function `ProMechshelladvancedMaterialorientGet()` returns the material orientations assigned to the advanced shell. Use the function `ProMechshelladvancedMaterialorientSet()` to set the material orientation for the advanced shell.

The function `ProMechshelladvancedShellpropsGet()` returns the shell properties associated with the model. Use the function `ProMechshelladvancedShellpropsSet()` to set the shell properties associated with the model.

Shell Properties

Shell properties are used to create shells that are not homogeneous, or to create shells that are comprised of several layers. A shell property can be assigned to a face, region, or datum surface.

The functions described in this section provide read and write access to data and contents of the Creo Simulate shell property objects. Shell properties use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_SHELL_PROPS`.

Functions Introduced:

- **ProMechshellpropsTypeGet()**
- **ProMechshellpropsDescriptionGet()**
- **ProMechshellpropsDescriptionSet()**
- **ProMechshellpropsHomogeneousdataGet()**
- **ProMechshellpropsHomogeneousdataSet()**
- **ProMechshellpropsLaminatelayupdataGet()**
- **ProMechshellpropsLaminatelayupdataSet()**

- **ProMechshellpropsLaminatestiffdataGet()**
- **ProMechshellpropsLaminatestiffdataSet()**
- **ProMechshlprophomogeneousAlloc()**
- **ProMechshlprophomogeneousThicknessGet()**
- **ProMechshlprophomogeneousThicknessSet()**
- **ProMechshlprophomogeneousFree()**
- **ProMechshlproplaminatestiffAlloc()**
- **ProMechshlproplaminatestiffAppliedstressGet()**
- **ProMechshlproplaminatestiffAppliedstressSet()**
- **ProMechshlproplaminatestiffBendingstiffnessGet()**
- **ProMechshlproplaminatestiffBendingstiffnessSet()**
- **ProMechshlproplaminatestiffCouplingstiffnessGet()**
- **ProMechshlproplaminatestiffCouplingstiffnessSet()**
- **ProMechshlproplaminatestiffExtensionalstiffnessGet()**
- **ProMechshlproplaminatestiffExtensionalstiffnessSet()**
- **ProMechshlproplaminatestiffIntertiaperunitareaGet()**
- **ProMechshlproplaminatestiffIntertiaperunitareaSet()**
- **ProMechshlproplaminatestiffMassperunitareaGet()**
- **ProMechshlproplaminatestiffMassperunitareaSet()**
- **ProMechshlproplaminatestiffTansverseshearGet()**
- **ProMechshlproplaminatestiffTansverseshearSet()**
- **ProMechshlproplaminatestiffThermalresforceGet()**
- **ProMechshlproplaminatestiffThermalresforceSet()**
- **ProMechshlproplaminatestiffThermalresmomentGet()**
- **ProMechshlproplaminatestiffThermalresmomentSet()**
- **ProMechshlproplaminatestiffFree()**
- **ProMechstresscalcdataAlloc()**
- **ProMechstresscalcdataCzGet()**
- **ProMechstresscalcdataCzSet()**
- **ProMechstresscalcdataMaterialIdGet()**
- **ProMechstresscalcdataMaterialIdSet()**
- **ProMechstresscalcdataPlyorientationGet()**
- **ProMechstresscalcdataPlyorientationSet()**

- **ProMechstresscalcdataFree()**
- **ProMechstresscalcdataProarrayFree()**
- **ProMechstiffmatrixEntry11Get()**
- **ProMechstiffmatrixEntry11Set()**
- **ProMechstiffmatrixEntry12Get()**
- **ProMechstiffmatrixEntry12Set()**
- **ProMechstiffmatrixEntry16Get()**
- **ProMechstiffmatrixEntry16Set()**
- **ProMechstiffmatrixEntry22Get()**
- **ProMechstiffmatrixEntry22Set()**
- **ProMechstiffmatrixEntry26Get()**
- **ProMechstiffmatrixEntry26Set()**
- **ProMechstiffmatrixEntry66Get()**
- **ProMechstiffmatrixEntry66Set()**
- **ProMechstiffmatrixFree()**
- **ProMechtransverseshearEntry44Get()**
- **ProMechtransverseshearEntry44Set()**
- **ProMechtransverseshearEntry45Get()**
- **ProMechtransverseshearEntry45Set()**
- **ProMechtransverseshearEntry55Get()**
- **ProMechtransverseshearEntry55Set()**
- **ProMechtransverseshearFree()**
- **ProMechthermalrescoeffEntry11Get()**
- **ProMechthermalrescoeffEntry11Set()**
- **ProMechthermalrescoeffEntry12Get()**
- **ProMechthermalrescoeffEntry12Set()**
- **ProMechthermalrescoeffEntry22Get()**
- **ProMechthermalrescoeffEntry22Set()**
- **ProMechthermalrescoeffFree()**
- **ProMechshlproplaminatelayupAlloc()**
- **ProMechshlproplaminatelayupLayersGet()**
- **ProMechshlproplaminatelayupLayersSet()**
- **ProMechshlproplaminatelayupTypeGet()**

- **ProMechshlproplaminatelayupTypeSet()**
- **ProMechshlproplaminatelayupFree()**
- **ProMechshlproplamlayuplayerMaterialIdGet()**
- **ProMechshlproplamlayuplayerMaterialIdSet()**
- **ProMechshlproplamlayuplayerNumberGet()**
- **ProMechshlproplamlayuplayerNumberSet()**
- **ProMechshlproplamlayuplayerOrientationGet()**
- **ProMechshlproplamlayuplayerOrientationSet()**
- **ProMechshlproplamlayuplayerThicknessGet()**
- **ProMechshlproplamlayuplayerThicknessSet()**
- **ProMechshlproplamlayuplayerShellpropsGet()**
- **ProMechshlproplamlayuplayerShellpropsSet()**
- **ProMechshlproplamlayuplayerFree()**
- **ProMechshlproplamlayuplayerProarrayFree()**

The function `ProMechshellpropsTypeGet()` returns the type of properties that have been defined for the shell. The types of shell properties are:

- `PRO_MECH_SHLPROP_HOMOGENEOUS`—Assigned to homogenous shells. A homogeneous shell consists of a single material whose properties do not vary through the thickness of the shell. Use the function `ProMechshellpropsHomogeneousdataGet()` to access the data structure for homogeneous shell property. Use the function `ProMechshellpropsHomogeneousdataSet()` to set the homogeneous shell property.
- `PRO_MECH_SHLPROP_LAMINATE_STIFFNESS`—Assigned to laminate shells to specify their degree of stiffness. Laminate shells consists of one or more materials whose properties may vary through the thickness of the shell. Use the function `ProMechshellpropsLaminatestiffdataGet()` to access the data structure for the laminate stiffness shell property. Use the function `ProMechshellpropsLaminatestiffdataSet()` to set the laminate stiffness shell property.
- `PRO_MECH_SHLPROP_LAMINATE_LAYUP`—Assigned to laminate shells to define them as layers of shells. Use the function `ProMechshellpropsLaminatelayupdataGet()` to access the data structure for the laminate layup shell property. Use the function `ProMechshellpropsLaminatelayupdataSet()` to set the laminate layup shell property.

The function `ProMechshellpropsDescriptionGet()` returns the description for the shell properties. Use the function `ProMechshellpropsDescriptionSet()` to modify the description for the shell properties.

The function `ProMechshlprophomogeneousThicknessGet()` returns the shell thickness defined for shells. Use the function `ProMechshlprophomogeneousThicknessSet()` to set the thickness for the shell properties.

The function `ProMechshlproplaminatestiffAppliedstressGet()` returns the calculation of stresses and strains for the laminate stiffness shell property type. The resultant array contains the shell "Top" location and the shell "Bottom" location. The values specified in these areas are used to calculate the stresses and strains for the corresponding areas. Use the function `ProMechshlproplaminatestiffAppliedstressSet()` to set the value of the stress and strain in the calculation array.

The function `ProMechstresscalcdataCzGet()` returns the distance from the midsurface of the shell at which the stresses and strains for the laminate shell is calculated. The Cz is defined relative to material direction 3 of the material orientation assigned to the shell. Use the function `ProMechstresscalcdataCzSet()` to set the Cz value used to calculate the stress and strain.

The function `ProMechstresscalcdataMaterialIdGet()` returns the material specified at the CZ location. Use the function `ProMechstresscalcdataMaterialIdSet()` to set the material id.

The function `ProMechstresscalcdataPlyorientationGet()` returns the orientation of the ply material. The ply orientation angle is measured as a counter-clockwise rotation from material direction 1 about material direction 3. Use the function `ProMechstresscalcdataPlyorientationSet()` to set the orientation of the ply material.

The function `ProMechshlproplaminatestiffBendingstiffnessGet()` returns the shell bending stiffness matrix. Use the function `ProMechshlproplaminatestiffBendingstiffnessSet()` to set the shell bending stiffness matrix.

The function `ProMechshlproplaminatestiffCouplingstiffnessGet()` returns the shell coupling stiffness matrix. Use the function `ProMechshlproplaminatestiffCouplingstiffnessSet()` to set the shell coupling stiffness matrix.

The function `ProMechshlproplaminatestiffExtensionalstiffnessGet()` returns the shell extensional stiffness matrix. Use the function `ProMechshlproplaminatestiffExtensionalstiffnessSet()` to set the shell extensional stiffness matrix.

The functions `ProMechstiffmatrixEntry11Get()`, `ProMechstiffmatrixEntry12Get()`, `ProMechstiffmatrixEntry16Get()`, `ProMechstiffmatrixEntry22Get()`, `ProMechstiffmatrixEntry26Get()`, and `ProMechstiffmatrixEntry66Get()` provide access the elements of the stiffness matrix.

Use the functions `ProMechstiffmatrixEntry11Set()`, `ProMechstiffmatrixEntry12Set()`, `ProMechstiffmatrixEntry16Set()`, `ProMechstiffmatrixEntry22Set()`, `ProMechstiffmatrixEntry26Set()`, and `ProMechstiffmatrixEntry66Set()` to modify the elements of the stiffness matrix.

The function `ProMechshlproplaminatestiffIntertiaperunitareaGet()` returns the rotary inertia per unit area for the laminate stiffness properties. Use the function `ProMechshlproplaminatestiffIntertiaperunitareaSet()` to set the inertia per unit area for the laminate stiffness properties.

The function `ProMechshlproplaminatestiffMassperunitareaGet()` returns the mass per unit area for the laminate stiffness properties. Use the function `ProMechshlproplaminatestiffMassperunitareaSet()` to set the mass per unit area for the laminate stiffness properties.

The function `ProMechshlproplaminatestiffTansverseshearGet()` returns the transverse sheer stiffness for the laminate shell. Use the function `ProMechshlproplaminatestiffTansverseshearSet()` to set the transverse sheer stiffness for the laminate shell.

The functions `ProMechtransverseshearEntry44Get()`, `ProMechtransverseshearEntry45Get()`, and `ProMechtransverseshearEntry55Get()` provide access to the elements 44, 45, and 55 of the transverse shear matrix. Use the functions `ProMechtransverseshearEntry44Set()`, `ProMechtransverseshearEntry45Set()`, and `ProMechtransverseshearEntry55Set()` to set the elements 44, 45, and 55 of the transverse shear matrix.

The functions `ProMechshlproplaminatestiffThermalresforceGet()` and `ProMechshlproplaminatestiffThermalresmomentGet()` return the thermal resultant coefficients for the laminate shell. The thermal coefficients are specified as Force and Moment. Use the functions `ProMechshlproplaminatestiffThermalresforceSet()` and `ProMechshlproplaminatestiffThermalresmomentSet()` to set the thermal resultant coefficients for the laminate shell.

The functions `ProMechthermalrescoeffEntry11Get()`, `ProMechthermalrescoeffEntry12Get()`, and, `ProMechthermalrescoeffEntry22Get()` provide access to the elements 11, 12, and 22 of the thermal resultant coefficient matrix. Use the functions `ProMechthermalrescoeffEntry11Set()`, `ProMechthermalrescoeffEntry12Set()`, `ProMechthermalrescoeffEntry22Set()` to set the elements 11, 12, and 22 of the thermal resultant coefficient matrix.

The function `ProMechshlproplaminatelayupLayersGet()` returns an array of layers, or plies, stacked on each other to form the laminate. Use the function `ProMechshlproplaminatelayupLayersSet()` to set the layers assigned to the laminate shell properties.

The function `ProMechshlproplaminatelayupTypeGet()` returns the layer repetition pattern for the laminate layup type shells. The types of repetition patterns are:

- `PRO_MECH_LAMLAYUP_SYMMETRIC`—Specifies that the layers are repeated in reverse order.
- `PRO_MECH_LAMLAYUP_ANTISYMMETRIC`—Specifies that the layers are repeated in reverse order, and the orientation is also changed.
- `PRO_MECH_LAMLAYUP_NEITHER`—Specifies that the layers are not repeated.

Use the function `ProMechshlproplaminatelayupTypeSet()` to set the type of laminate layup shell properties.

The function `ProMechshlproplamlayuplayerMaterialIdGet()` returns the name of the material assigned to the specified layer Use the function `ProMechshlproplamlayuplayerMaterialIdSet()` to set the id of the specified laminate layer.

The function `ProMechshlproplamlayuplayerNumberGet()` returns the number of times a particular layer is repeated for the laminate. Use the function `ProMechshlproplamlayuplayerNumberSet()` to set the number for the laminate layup layer.

The function `ProMechshlproplamlayuplayerOrientationGet()` returns the orientation of the specified layer of the laminate. Use the function `ProMechshlproplamlayuplayerOrientationSet()` to set the orientation of the specified layer of the laminate.

The function `ProMechshlproplamlayuplayerThicknessGet()` returns the thickness of the specified layer of the laminate. Use the function `ProMechshlproplamlayuplayerThicknessSet()` to set the thickness of the specified layer of the laminate.

The function `ProMechshlproplamlayuplayerShellpropsGet()` returns the shell properties for the specified laminate layer. Use the function `ProMechshlproplamlayuplayerShellpropsSet()` to set the shell properties.

Shell Pairs

The shell pairs are created based on the surfaces belonging to a part. The shell pairs are compressed to a mid surface or set of mid surfaces and shell elements are assigned to it.

The functions described in this section provide read and write access to the data and contents of the Creo Simulate shell pair objects. Shell pairs use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_SHELL_PAIR`.

Functions Introduced:

- **ProMechshlpairrefsAlloc()**
- **ProMechshellpairMaterialIdGet()**
- **ProMechshellpairMaterialIdSet()**
- **ProMechshellpairMaterialOrientIdGet()**
- **ProMechshellpairMaterialOrientIdSet()**
- **ProMechshellpairReferencesGet()**
- **ProMechshellpairReferencesSet()**
- **ProMechshlpairrefsTypeGet()**
- **ProMechshlpairrefsTypeSet()**
- **ProMechshlpairrefsPlacementtypeGet()**
- **ProMechshlpairrefsPlacementtypeSet()**
- **ProMechshlpairrefsBottomreferencesGet()**
- **ProMechshlpairrefsBottomreferencesSet()**
- **ProMechshlpairrefsTopreferencesGet()**
- **ProMechshlpairrefsTopreferencesSet()**

- **ProMechshlpairrefsSelectedplacementGet()**
- **ProMechshlpairrefsSelectedplacementSet()**
- **ProMechshlpairrefsExtendAdjacentSurfacesGet()**
- **ProMechshlpairrefsExtendAdjacentSurfacesSet()**
- **ProMechshlpairrefsFree()**

The function `ProMechshellpairMaterialIdGet()` returns the material used to create the shell pair. Use the function `ProMechshellpairMaterialIdSet()` to set the material id of the shell pair.

The function `ProMechshellpairReferencesGet()` returns the geometric references for the shell pair item. Use the function `ProMechshellpairReferencesSet()` to set the geometric references for the shell pair item.

The function `ProMechshlpairrefsTypeGet()` returns the type of shell pair references. The types of shell pairs are:

- `PRO_MECH_SHELL_PAIR_CONSTANT`—Specifies a shell pair with constant thickness. All opposing surfaces parallel to each other and equidistant from the opposing surface for a constant-thickness shell pair.
- `PRO_MECH_SHELL_PAIR_VARIABLE`—Specifies a variable thickness shell pair. Both opposing surfaces are neither parallel nor concentric for a variable thickness shell pair.

> **Note**
>
> From Creo Parametric onwards, the shell type `PRO_MECH_SHELL_PAIR_VARIABLE` is also supported in the Native mode of Creo Simulate .

- `PRO_MECH_SHELL_PAIR_MULTI_CONSTANT`—Specifies a shell pair with multiple pairs of surfaces. For a pair of surfaces, each surface is parallel to and equidistant from the opposing surface. However, the distance between the surfaces for each of the multiple pairs may vary.

Note

From Creo Parametric onwards, the thickness type `PRO_MECH_SHELL_PAIR_MULTI_CONSTANT` has been deprecated and the function `ProMechshlpairrefsTypeSet()` will return an error type `PRO_TK_UNSUPPORTED`.

Use the function `ProMechshlpairrefsTypeSet()` to set the type of shell pair references.

The functions `ProMechshlpairrefsTopreferencesGet()` and `ProMechshlpairrefsBottomreferencesGet()` provide access to the top and bottom references for the shell pair. This should be defined for all shell pair types. Use the functions `ProMechshlpairrefsBottomreferencesSet()` and `ProMechshlpairrefsTopreferencesSet()` to set the top and bottom references for the shell pair.

The function `ProMechshlpairrefsPlacementtypeGet()` returns the placement of shell pair. The types of placement references are as follows:

- `PRO_MECH_SHELL_PAIR_PLACEMENT_TOP`—Specifies that the placement uses the top surface of the surface pair.
- `PRO_MECH_SHELL_PAIR_PLACEMENT_BOTTOM`— Specifies that the placement uses the bottom surface of the surface pair.
- `PRO_MECH_SHELL_PAIR_PLACEMENT_MIDDLE`— Specifies that the placement uses the mid surface of the surface pair.
- — Specifies that the placement uses an arbitrary selected surface, which can be a datum surface.

Use the function `ProMechshlpairrefsPlacementtypeSet()` to set the placement type of the shell pair.

Use the function `ProMechshlpairrefsSelectedplacementGet()` to access the data structure containing the selected surface. Use the function `ProMechshlpairrefsSelectedplacementSet()` to set the surface for the placement reference of type SELECTED.

Use the function `ProMechshlpairrefsExtendAdjacentSurfacesGet()` to retrieve the value of the flag that indicates whether the adjacent surfaces in a shell pair will be extended during meshing or not. For a mixed model, you can set this flag to extend surfaces adjacent to the top and bottom surfaces of the shell pair by using the function `ProMechshlpairrefsExtendAdjacentSurfacesSet()`. However, this is possible only if the angle between the shell pair surfaces and their adjacent surfaces is less than the value specified by the configuration option.

Note

For a midsurface model, Creo Simulate extends the adjacent surfaces regardless of the value of `sim_extend_surf_max_angle`. The default value of `sim_extend_surf_max_angle` is 30 degrees.

You must initialize the Creo Simulate environment to use these functions. For shell pairs defined in Pro/ENGINEER Mechanica 4.0 and earlier, this flag is set to 1 by default.

Use the function `ProMechshlpairrefsFree()` to free the simple shell data handle in Creo Simulate .

Interfaces

Interfaces, also called connections, are used to connect surfaces. When you create an interface in Creo Simulate , specify how Creo Simulate will treat the connected surfaces during meshing and analysis. Interfaces use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_INTERFACE`.

Functions Introduced:

- **ProMechinterfaceTypeGet()**
- **ProMechinterfaceReferencesGet()**
- **ProMechinterfaceReferencesSet()**
- **ProMechinterfacebonddataAlloc()**
- **ProMechinterfaceBonddataGet()**
- **ProMechinterfaceBonddataSet()**
- **ProMechinterfacebonddataFree()**
- **ProMechinterfacefreedataAlloc()**
- **ProMechinterfaceFreedataGet()**
- **ProMechinterfaceFreedataSet()**
- **ProMechinterfacefreedataFree()**
- **ProMechinterfacecontactdataAlloc()**
- **ProMechinterfaceContactdataGet()**
- **ProMechinterfaceContactdataSet()**
- **ProMechinterfacecontactdataFree()**
- **ProMechinterfacethrresistdataAlloc()**

- **ProMechinterfaceThrresistdataGet()**
- **ProMechinterfaceThrresistdataSet()**
- **ProMechinterfacethrresistdataFree()**

The function `ProMechinterfaceTypeGet()` returns the type of interface used to connect surfaces. The types of interfaces are:

- Structural—Specifies the default for the interfaces created between the geometry in a structural model for meshing and running.
  - `PRO_MECH_INTERFACE_BOND_STRUCT`—Specifies that the contacting surfaces as bonded. This means that matching nodes on contacting surfaces merge.
  - `PRO_MECH_INTERFACE_CONTACT_STRUCT`—Specifies an interface between components when you want the components to have the freedom to remain separate from each other, and also when you want the components to transfer forces between them when they touch, or come into contact with each other.
  - `PRO_MECH_INTERFACE_FREE_STRUCT`—Specifies two surfaces as contacting, but does not merge the nodes. Meshes on the contacting surfaces are identical, and matching nodes are coincident.
- Thermal—Specifies the default for the interfaces created between the geometry in a thermal model for meshing and running.
  - `PRO_MECH_INTERFACE_BOND_THERM`—Specifies that coincident geometry is bonded.
  - `PRO_MECH_INTERFACE_FREE_THERM`—Specifies that no geometry in the assembly is merged.
  - `PRO_MECH_INTERFACE_RESIST_THERM`—Specify this interface type to create a thermal resistance interfaces at run-time.

The function `ProMechinterfaceReferencesGet()` returns the geometric entities selected to create the interface. Use the function `ProMechinterfaceReferencesSet()` to set the geometric entities selected to create the interface.

The function `ProMechinterfacebonddataAlloc()` allocates memory for the Creo Simulate bonded interface data.

The function `ProMechinterfaceBonddataGet()` provides access to the bonded interface data. The function returns whether the specified bonded interface should use links between pairs of nodes.

Use the function `ProMechinterfaceBonddataSet()` to assign the bonded interface data to the Creo Simulate item.

Use the function `ProMechinterfacebonddataFree()` to free the memory of the bonded interface data.

The function `ProMechinterfacefreedataAlloc()` allocates memory for the Creo Simulate free interface data structure.

The function `ProMechinterfaceFreedataGet()` provides access to the free interface data. The function returns a flag that indicates whether the meshes generated on the surfaces of the interface are coincident or not. Use the function `ProMechinterfaceFreedataSet()` to assign the free interface data to the Creo Simulate item.

The function `ProMechinterfacecontactdataAlloc()` allocates memory for the Creo Simulate contact interface data structure.

The function `ProMechinterfaceContactdataGet()` provides access to the properties of the contact interface data. Use the function `ProMechinterfaceContactdataSet()` to assign the contact interface data to the Creo Simulate item.

The function `ProMechinterfacethrresistdataAlloc()` allocates memory for the Creo Simulate thermal resistance interface data structure.

The function `ProMechinterfaceThrresistdataGet()` provides access to the thermal resistance interface data. Use the function `ProMechinterfaceThrresistdataSet()` to assign the thermal resistance interface data to the Creo Simulate item.

Bonded Interface

Functions Introduced:

- **ProMechbondinterfacedataMergenodesGet()**
- **ProMechbondinterfacedataMergenodesSet()**

The function `ProMechbondinterfacedataMergenodesGet()` specifies if coincident nodes of components or surfaces touching each other are merged during meshing. Use the function `ProMechbondinterfacedataMergenodesSet()` to set whether coincident nodes for bonded interface should merge.

Contact Interface

Functions Introduced:

- **ProMechcontactinterfacedataSeparationdistanceExprGet()**
- **ProMechcontactinterfacedataSeparationdistanceExprSet()**
- **ProMechcontactinterfacedataAnglebetweensurfacesExprGet()**
- **ProMechcontactinterfacedataAnglebetweensurfacesExprSet()**

- **ProMechcontactinterfacedataCheckonlyplanarGet()**
- **ProMechcontactinterfacedataCheckonlyplanarSet()**
- **ProMechcontactinterfacedataSlippageGet()**
- **ProMechcontactinterfacedataSlippageSet()**
- **ProMechcontactinterfacedataCoefffrictionGet()**
- **ProMechcontactinterfacedataCoefffrictionSet()**
- **ProMechcontactinterfacedataSplitsurfacesGet()**
- **ProMechcontactinterfacedataSplitsurfacesSet()**
- **ProMechcontactinterfacedataCompatiblemeshGet()**
- **ProMechcontactinterfacedataCompatiblemeshSet()**
- **ProMechcontactinterfacedataFrictionSet()**
- **ProMechcontactinterfacedataFrictionGet()**
- **ProMechcontactinterfacedataDynamicCoefffrictionSet()**
- **ProMechcontactinterfacedataDynamicCoefffrictionGet()**
- **ProMechcontactinterfacedataDynamicCoeffSameAsStaticSet()**
- **ProMechcontactinterfacedataDynamicCoeffSameAsStaticGet()**
- **ProMechcontactinterfacedataUseSelectionFilterTolSet()**
- **ProMechcontactinterfacedataUseSelectionFilterTolGet**

To create a contact interface between two components of an assembly, with the reference type as component-component, specify the selection filter tolerance criteria. You can specify the selection filter tolerance criteria using the following functions.

Use the function `roMechcontactinterfacedataSeparationdistanceExprGet()` to retrieve the separation distance. Separation distance is the distance between the surface pairs that you want to use to define a contact interface. This separation distance is the limit beyond which a contact interface cannot be created. If the distance between the surfaces of two components is smaller than the separation distance, the surfaces are used for creation of the contact interface.

The separation distance is returned as an expression of type `ProMechExpression`.

Note

The function `ProMechcontactinterfacedataSeparationdistanceExprGet()` supersedes the function `ProMechcontactinterfacedataSeparationdistanceGet()`.

The function `ProMechcontactinterfacedataSeparationdistanceExprSet()` sets the separation distance. Specify this distance as an expression of type `ProMechExpression`.

Note

The function `ProMechcontactinterfacedataSeparationdistanceExprSet()` supersedes the function `ProMechcontactinterfacedataSeparationdistanceSet()`.

The function `ProMechcontactinterfacedataAnglebetweensurfacesExprGet()` returns the angle between planar surfaces while creating contact interface. This angle is returned as an expression of type `ProMechExpression`.

Note

The function `ProMechcontactinterfacedataAnglebetweensurfacesExprGet()` supersedes the function `ProMechcontactinterfacedataAnglebetweensurfacesGet()`.

The function `ProMechcontactinterfacedataAnglebetweensurfacesExprSet()` sets the angle between surfaces while creating contact interface. Specify this angle as an expression of type `ProMechExpression`.

Note

The function `ProMechcontactinterfacedataAnglebetweensurfacesExpr Set()` supersedes the function `ProMechcontactinterfacedataAnglebetweensurfacesSet()`.

The function `ProMechcontactinterfacedataCheckonlyplanarGet()` returns a true if the contact is created between planar surfaces in a component-component type interface.

Use the function `ProMechcontactinterfacedataCheckonlyplanarSet()` to create contacts only between planar surfaces.

The properties of the contact interface are as follows:

Use the function `ProMechcontactinterfacedataFrictionSet()` to set the type of the friction at the contact interface, between the pairs of nodes, using the enumerated type `ProMechInterfaceFrictionType`. The valid values are:

- `PRO_MECH_INTERFACE_FRICTION_INFINITE`—Specifies that infinite friction exists at the contact interface that is, the two components or surfaces cannot slide relative to each other.
- `PRO_MECH_INTERFACE_FRICTION_FINITE`—Specifies that finite friction exists at the contact interface that is, that is, the two components or surfaces are able to slide relative to each other.

Use the function `ProMechcontactinterfacedataFrictionGet()` to obtain the type of friction present at the contact interface.

The function `ProMechcontactinterfacedataSlippageGet()` returns true if slippage has occurred in the contact region during the analysis. Use the function `ProMechcontactinterfacedataSlippageSet()` to check for slippage in the contact area during analysis.

The function `ProMechcontactinterfacedataCoefffrictionGet()` returns the coefficient of friction used to calculate the slippage in the contact region during analysis. Use the function `ProMechcontactinterfacedataCoefffrictionSet()` to set the coefficient of friction for computing the slippage. Specify a positive value as the coefficient of friction.

The function `ProMechcontactinterfacedataSplitsurfacesGet()` returns true if you split the surface shared by the volumes used to define the interface.

Use the function `ProMechcontactinterfacedataSplitsurfacesSet()` to specify whether the interface should split surfaces.

The function `ProMechcontactinterfacedataCompatiblemeshGet()` specifies if a compatible mesh is created when components in the assembly are touching.

Use the function `ProMechcontactinterfacedataCompatiblemeshSet()` to create geometrically-consistent node locations when the mesh for the surfaces of your interface is generated.

Use the function `ProMechcontactinterfacedataDynamicCoefffrictionSet()` to set the dynamic coefficient of friction for the contact interface. The dynamic coefficient of friction prevents the axis surfaces from moving freely against each other which slows down the motion.

Note

Set a value less than or equal to the value of static coefficient of friction for this interface.

Use the function `ProMechcontactinterfacedataDynamicCoefffrictionGet()` to obtain the dynamic coefficient of friction for the specified contact interface data. Use the function `ProMathExpressionEvaluate()` to evaluate the dynamic coefficient of friction.

Use the function `ProMechcontactinterfacedataDynamicCoeffSameAsStatic Set()` to set the dynamic coefficient of friction same as the static coefficient of friction. Pass the value `PRO_B_TRUE` to set the dynamic coefficient of friction same as static coefficient of friction.

Use the function `ProMechcontactinterfacedataDynamicCoeffSameAsStatic Get()` to identify whether the dynamic coefficient of friction is same as the static coefficient of friction. This function returns the value `PRO_B_TRUE` if the dynamic and static coefficient of friction possess the same value.

Use the function `ProMechcontactinterfacedataUseSelectionFilterTolSet()` to set the selection filter tolerance between contact surfaces. Pass the value `PRO_B_ TRUE` to switch on the selection filter tolerances.

Use the function `ProMechcontactinterfacedataUseSelectionFilterTolGet()` to identify whether the specified contact interface uses the selection filter tolerances between the contact surfaces. This function returns the value `PRO_B_TRUE` if the contact surfaces use the selection filter tolerances.

Thermal Resistance Interface

Functions Introduced:

- **ProMechthrresistinterfacedataConductivityGet()**
- **ProMechthrresistinterfacedataConductivitySet()**

The function `ProMechthrresistinterfacedataConductivityGet()` returns the heat transfer coefficient for the interface. The heat transfer coefficient is the heat that passes through the interface per unit time per unit area when the temperature difference between opposite interface surfaces is one unit.

Use the function `ProMechthrresistinterfacedataConductivitySet()` to set the heat transfer coefficient for the interface. Specify a positive number as the heat transfer coefficient.

Free Interface

Functions Introduced:

- **ProMechfreeinterfacedataSplitsurfacesGet()**
- **ProMechfreeinterfacedataSplitsurfacesSet()**

The function `ProMechfreeinterfacedataSplitsurfacesGet()` returns true if you split the surface shared by the volumes used to define the free interface.

The function `ProMechfreeinterfacedataSplitsurfacesSet()` to specify whether the free interface should split surfaces.

Gaps

A gap is a nonlinear element, used to model connection between points, edges and curves, or surfaces in your model by connecting two nodes in separated geometries. Gaps use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_GAP`.

Functions Introduced:

- **ProMechgapTypeGet()**
- **ProMechgapReferencesGet()**

- **ProMechgapReferencesSet()**
- **ProMechgapSimpledataGet()**
- **ProMechgapSimpledataSet()**
- **ProMechsimplegapdataAlloc()**
- **ProMechsimplegapdataFree()**
- **ProMechsimplegapdataYdirectionGet()**
- **ProMechsimplegapdataYdirectionSet()**
- **ProMechsimplegapdataDistributiontypeGet()**
- **ProMechsimplegapdataDistributiontypeSet()**
- **ProMechsimplegapdataAxialstiffnessGet()**
- **ProMechsimplegapdataAxialstiffnessSet()**
- **ProMechsimplegapdataTransversestiffnessGet()**
- **ProMechsimplegapdataTransversestiffnessSet()**
- **ProMechsimplegapdataClearanceGet()**
- **ProMechsimplegapdataClearanceSet()**

The function `ProMechgapTypeGet()` returns the type of gap for the specified Creo Simulate Gap.

The function `ProMechgapReferencesGet()` returns the geometric entities of the model that are selected to create the gap. Use the function `ProMechgapReferencesSet()` to set the valid geometric references for the specified gap.

The function `ProMechgapSimpledataGet()` returns the data structure for the Creo Simulate gap data. The Creo Simulate gap data structure defines the y-direction and the stiffness properties of the gap. Use the function `ProMechgapSimpledataSet()` to set the Creo Simulate gap data structure.

The function `ProMechsimplegapdataAlloc()` allocates memory for the Creo Simulate gap data structure.

`ProMechsimplegapdataYdirectionGet()` returns the orientation of the XY-plane of the gap. Use the function `ProMechsimplegapdataYdirectionSet()` to set the Y-direction of the gap data.

The function `ProMechsimplegapdataDistributiontypeGet()` returns the method used to calculate the axial and transverse stiffnesses for the gap data. The types of distribution are as follows:

- PRO_MECH_GAP_DISTR_TOTAL—Specifies the sum of stiffness of all the contact elements.
- PRO_MECH_GAP_DISTR_PER_UNIT—Specifies that the stiffness value is calculated using the area of the first selected surface.

Use the function ProMechsimplegapdataDistributiontypeSet() to set the distribution type for the gap data.

The function ProMechsimplegapdataAxialstiffnessGet() returns the axial stiffness for the gap. The axial stiffness defines a stiffness or spring factor. Use the function ProMechsimplegapdataAxialstiffnessSet() to set the axial stiffness for the gap.

The function ProMechsimplegapdataTransversestiffnessGet() returns the transverse stiffness for the gap. The transverse stiffness defines the elastic stiffness of the material. Use the function ProMechsimplegapdataTransversestiffnessSet() to set the transverse stiffness of the gap.

The function ProMechsimplegapdataClearanceGet() returns the distance between two nodes at which the axial and transverse stiffnesses are active due to displacement during analysis. Use the function ProMechsimplegapdataClearanceSet() to set the clearance for the gap data.

Mesh Control

Mesh controls define the characteristics of a mesh. Mesh controls specify the minimum or maximum size of the elements, the distribution of nodes along edges, hard points and hard curves, mesh ID numbering, mesh ID offsets, and the displacement coordinate system. The functions described in this section provide access to the data and contents of the Creo Simulate mesh control objects. Mesh controls use the ProType field in the ProMechitem structure as PRO_SIMULATION_MESH_CNTRL.

Functions Introduced:

- **ProMechmeshcntrlTypeGet()**
- **ProMechmeshcntrlAutogemedgedistrdataGet()**
- **ProMechmeshcntrlAutogemedgedistrdataSet()**
- **ProMechmeshcntrlAutogemminedgedataGet()**
- **ProMechmeshcntrlAutogemminedgedataSet()**
- **ProMechmeshcntrlAutogemelemsizedataGet()**
- **ProMechmeshcntrlAutogemelemsizedataSet()**
- **ProMechmeshcntrlAutogemedgelencrvdataGet()**

- **ProMechmeshcntrlAutogemedgelencrvdataSet()**
- **ProMechmeshcntrlEdgedistrdataGet()**
- **ProMechmeshcntrlEdgedistrdataSet()**
- **ProMechmeshcntrlElemsizedataGet()**
- **ProMechmeshcntrlElemsizedataSet()**
- **ProMechmeshcntrlShellcsysGet()**
- **ProMechmeshcntrlShellcsysSet()**
- **ProMechmeshcntrlHardpointGet()**
- **ProMechmeshcntrlHardpointSet()**
- **ProMechmeshcntrlHardcurveSet()**
- **ProMechmeshcntrlIdsoffsetGet()**
- **ProMechmeshcntrlIdsoffsetSet()**
- **ProMechmeshcntrlNumberingGet()**
- **ProMechmeshcntrlNumberingSet()**
- **ProMechmeshcntrlSuppressGet()**
- **ProMechmeshcntrlSuppressSet()**
- **ProMechmcautogemsuppressTypeGet()**
- **ProMechmcautogemsuppressTypeSet()**
- **ProMechmeshcntrlAutogemsuppressGet()**
- **ProMechmeshcntrlAutogemsuppressSet()**
- **ProMechmeshcntrlAutogemisolateexcludedataGet()**
- **ProMechmeshcntrlAutogemisolateexcludedataSet()**
- **ProMechmeshcntrlReferencesGet()**
- **ProMechmeshcntrlReferencesSet()**
- **ProMechmcautogemHardpointSet()**
- **ProMechmcautogemHardcurveSet()**

The function `ProMechmeshcntrlTypeGet()` returns the type of mesh control. The types of mesh controls are:

- `PRO_MECH_MC_AGEM_EDGE_DISTR`—Specifies the edge distribution data for AutoGEM. Use the function `ProMechmeshcntrlAutogemedgedistrdataGet()` to access the AutoGEM edge distribution data for the mesh control item. Use the function `ProMechmeshcntrlAutogemedgedistrdataSet()` to modify the AutoGEM edge distribution data for the mesh control item.

Note

If you are creating a new mesh control using this type of data, you should assign the model as a reference using the function `ProMechmeshcntrlReferencesSet()`.

- `PRO_MECH_MC_AGEM_MIN_EDGE`—Specifies the minimum edge and face angles for AutoGEM. Use the function `ProMechmeshcntrlAutogemminedgedataGet()` to access the AutoGEM minimum edge data for the mesh control item. Use the function `ProMechmeshcntrlAutogemminedgedataSet()` to modify the AutoGEM minimum edge data for the mesh control item.

Note

If you are creating a new mesh control using this type of data, you should assign the model as a reference using the function `ProMechmeshcntrlReferencesSet()`.

- `PRO_MECH_MC_AGEM_MAX_ELEMENT_SIZE`—Specifies the maximum element size for the AutoGEM mesh. Use the function `ProMechmeshcntrlAutogemelemsizedataGet()` to access the AutoGEM maximum element data for the mesh control item. Use the function `ProMechmeshcntrlAutogemelemsizedataSet()` to modify the AutoGEM maximum element data for the mesh control item.

Note

If you are creating a new mesh control using this type of data, you should assign the model as a reference using the function `ProMechmeshcntrlReferencesSet()`.

- `PRO_MECH_MC_AGEM_EDGE_LEN_CRV`—Specifies the ratio of edge lengths of mesh elements adjacent to concave surfaces to the radius of curvature of the concave surfaces. Use the function `ProMechmeshcntrlAutogemedgelencrvdataGet()` to access the edge length by curvature data for the mesh control item. Use the function `ProMechmeshcntrlAutogemedgelencrvdataSet()` to set the edge length by curvature ratio.

- PRO_MECH_MC_EDGE_DISTRIBUTION—Specifies the number of nodes on one or more edges of the curves in the model. Use the function ProMechmeshcntrlEdgedistrdataGet() to access the edge distribution data for the mesh control item. Use the function ProMechmeshcntrlEdgedistrdataSet() to set the edge distribution data.

> **Note**
>
> If you are creating a new mesh control using this type of data, you should assign the model as a reference using the function ProMechmeshcntrlReferencesSet().

- PRO_MECH_MC_DISPL_CSYS—Specifies the displacement coordinate system used for displaying results for nodes associated with points, edges, curves, or surfaces.
- PRO_MECH_MC_MAX_ELEMENT_SIZE—Specifies the maximum element size for the mesh. Use the function ProMechmeshcntrlElemsizedataGet() to access the maximum element size data for the mesh control item. Use the function ProMechmeshcntrlElemsizedataSet() to set the maximum element size for the mesh.

> **Note**
>
> If you are creating a new mesh control using this type of data, you should assign the model as a reference using the function ProMechmeshcntrlReferencesSet().

- PRO_MECH_MC_MIN_ELEMENT_SIZE—Specifies the minimum element size for the mesh. Use the function ProMechmeshcntrlElemsizedataGet() to access the minimum element size data for the mesh control item. Use the function ProMechmeshcntrlElemsizedataSet() to set the maximum element size for the mesh.
- PRO_MECH_MC_SHELL_CSYS—Specifies the coordinate system used for displaying results for nodes associated with shell or quilt surfaces. Use the function ProMechmeshcntrlShellcsysGet() to access the shell coordinate system for the mesh control item. Use the function ProMechmeshcntrlShellcsysSet() to set the shell coordinate system for the mesh control item.

- PRO_MECH_MC_HARD_POINT—Specifies a datum point that is defined as a hard point. Use the function ProMechmeshcntrlHardpointGet() to access the hard point data for the mesh control item. Use the function ProMechmeshcntrlHardpointSet() to set the hard point data for the mesh control item.
- PRO_MECH_MC_HARD_CURVE—Specifies a datum curve that is defined as a hard curve. This type has no data, only references. Use the function ProMechmeshcntrlHardcurveSet() to set the mesh control item to be of the type hard curve. The reference for the item will be used for the control.
- PRO_MECH_MC_IDS_OFFSET—Specifies the offset for the node IDs and element IDs for a component. Use the function ProMechmeshcntrlIdsoffsetGet() to access the offset data for the mesh control item. Use the function ProMechmeshcntrlIdsoffsetSet() to set the offset data.
- PRO_MECH_MC_MESH_NUMBERING—Specifies a node and an element ID range for a component. Use the function ProMechmeshcntrlNumberingGet() to access the numbering data for the mesh control item. Use the function ProMechmeshcntrlNumberingSet() to set the numbering data for the mesh control item.
- PRO_MECH_MC_SUPPRESS—Specifies the components that must be ignored while applying the mesh control to the assembly level. Use the function ProMechmeshcntrlSuppressGet() to access the suppress data for the mesh control item. Use the function ProMechmeshcntrlSuppressSet() to set the suppress data for the mesh control item.
- PRO_MECH_MC_AGEM_SUPPRESS—Specifies the type of AutoGEM controls that you want the mesh generator to ignore at the assembly level. Use the function ProMechmcautogemsuppressTypeGet() to access the types of AutoGEM control suppressed by this AutoGEM control data. Use the function ProMechmcautogemsuppressTypeSet() to set the type of AutoGEM control suppressed by this AutoGEM control data.

  The functions ProMechmeshcntrlAutogemsuppressGet() and ProMechmeshcntrlAutogemsuppressSet() access and set the suppress data for the AutoGEM control items.

  These functions support only the following AGEM mesh control types:

  - PRO_MECH_MC_AGEM_EDGE_DISTR
  - PRO_MECH_MC_AGEM_MIN_EDGE
  - PRO_MECH_MC_AGEM_ISOLATE_EXCLUDE

- PRO_MECH_MC_AGEM_MAX_ELEMENT_SIZE
- PRO_MECH_MC_AGEM_EDGE_LEN_CRV
- PRO_MECH_MC_AGEM_HARD_POINT
- PRO_MECH_MC_AGEM_HARD_CURVE
- PRO_MECH_MC_ALL

- PRO_MECH_MC_AGEM_ISOLATE_EXCLUDE—Specifies points, edges, curves, and surfaces from the model to isolate during analysis. The function ProMechmeshcntrlAutogemisolateexcludedataGet() returns the list of entities that AutoGEM can detect and isolate using mesh refinement. Use the function ProMechmeshcntrlAutogemisolateexcludedataSet() to set the entities for AutoGEM isolation using mesh refinement.
- PRO_MECH_MC_AGEM_HARD_POINT—Specifies points, point features, or point patterns on the model to guide the AutoGEM mesh creation process. This control type has only references and no data associated with it. Use the function ProMechmcautogemHardpointSet() to set the AutoGEM mesh control item as hard point.
- PRO_MECH_MC_AGEM_HARD_CURVE—Specifies the datum curves on the model to guide the AutoGEM mesh creation process. This control type has only references and no data associated with it. Use the function ProMechmcautogemHardcurveSet() to set the AutoGEM mesh control item as hard curve.
- PRO_MECH_MC_ALL—Specifies all the types of mesh controls.

Note

If you are creating a new mesh control using this type of data, you should assign the model as a reference using the function ProMechmeshcntrlReferencesSet().

The function ProMechmeshcntrlReferencesGet() returns the references for each of the mesh control type. Use the function ProMechmeshcntrlReferencesSet() to set the references used by the mesh control item.

Accessing AutoGEM Edge Distribution and Minimum Edge Mesh Control Data

The functions described in this section provide access to the data for the AutoGEM edge distribution Mesh Control.

Functions Introduced:

- **ProMechmcautogemedgedistrAlloc()**
- **ProMechmcautogemedgedistrNodesGet()**
- **ProMechmcautogemedgedistrNodesSet()**
- **ProMechmcautogemedgedistrRatioGet()**
- **ProMechmcautogemedgedistrRatioSet()**
- **ProMechmcautogemedgedistrStrictGet()**
- **ProMechmcautogemedgedistrStrictSet()**
- **ProMechmcautogemedgedistrFree()**
- **ProMechmcautogemminedgeAlloc()**
- **ProMechmcautogemminedgeEdgesGet()**
- **ProMechmcautogemminedgeEdgesSet()**
- **ProMechmcautogemminedgeLengthGet()**
- **ProMechmcautogemminedgeLengthSet()**
- **ProMechmcautogemminedgeFree()**

The function `ProMechmcautogemedgedistrNodesGet()` returns the number of nodes distributed along the selected edge. Use the function `ProMechmcautogemedgedistrNodesSet()` to set the number of nodes.

The function `ProMechmcautogemedgedistrRatioGet()` returns the aspect ratio defined while creating and editing elements. The aspect ratio is defined as the ratio of a length to the width of any surface in the model. Use the function `ProMechmcautogemedgedistrRatioSet()` to set the aspect ratio for the AutoGEM edge distribution mesh control data.

The function `ProMechmcautogemedgedistrStrictGet()` returns the value `true`, if the aspect ratio is within the maximum allowable range while creating and editing elements. Use the function `ProMechmcautogemedgedistrStrictSet()` to set the aspect ratio within the maximum allowable range.

Use the function `ProMechmcautogemedgedistrFree()` to free the memory containing the AutoGEM edge distribution data structure.

The function `ProMechmcautogemminedgeEdgesGet()` specifies the number of edges in a model for the AutoGEM minimum edge data. Use the function `ProMechmcautogemminedgeEdgesSet()` to set the number of edges.

The function `ProMechmcautogemminedgeLengthGet()` specifies the minimum length of any edge in the model. Use the function `ProMechmcautogemminedgeLengthSet()` to specify the minimum length of the edge for the AutoGEM min edge mesh control data.

Use the function `ProMechmcautogemminedgeFree()` to free the memory containing the AutoGEM minimum edge data structure.

Accessing the AutoGEM Edge Length by Curvature Mesh Control Data

The functions described in this section enable you to create a denser element mesh adjacent to areas such as curves, fillets, and holes, that are likely to have high stress, using the edge length by curve ratio.

- **ProMechmcautogemedgelencrvAlloc()**
- **ProMechmcautogemedgelencrvRatioGet()**
- **ProMechmcautogemedgelencrvRatioSet()**
- **ProMechmcautogemedgelencrvIgnoreRadiusGet()**
- **ProMechmcautogemedgelencrvIgnoreRadiusSet()**
- **ProMechmcautogemedgelencrvMinradiusGet()**
- **ProMechmcautogemedgelencrvMinradiusSet()**
- **ProMechmcautogemedgelencrvFree()**

The function `ProMechmcautogemedgelencrvAlloc()` allocates memory for the Creo Simulate mesh control edge length by curvature data handle.

The function `ProMechmcautogemedgelencrvRatioGet()` returns the ratio of the expected edge lengths of mesh elements to the radius of the concave surface. Use the function `ProMechmcautogemedgelencrvRatioSet()` to set this ratio. Specify the ratio as a positive real number.

The function `ProMechmcautogemedgelencrvMinradiusGet()` returns the cut-off value for the radius of curvature below which the AutoGEM mesh control data will not be applied. Use the function `ProMechmcautogemedgelencrvMinradiusSet()` to set the cut-off value for the radius of curvature. This value can be specified in the current units of the model or as a percentage of the model size.

`ProMechmcautogemedgelencrvIgnoreRadiusGet()` returns true, if the curves having a radius of curvature lower than the cut-off value must be ignored by the AutoGEM mesh control data. Use the function `ProMechmcautogemedgelencrvIgnoreRadiusSet()` to specify whether to ignore curves having a radius of curvature lower than the cut-off value.

Use the function `ProMechmcautogemedgelencrvFree()` to free the memory containing the edge length by curvature mesh control data structure.

Accessing the AutoGEM Maximum Element Size Mesh Control Data

The functions described in this section enable you to control the size of the elements created by the mesh generator for components, volumes, surfaces, edges, or curves.

Functions Introduced:

- **ProMechmcautogemelemsizeAlloc()**
- **ProMechmcautogemelemsizeSizeGet()**
- **ProMechmcautogemelemsizeSizeSet()**
- **ProMechmcautogemelemsizeFree()**

The function `ProMechmcautogemelemsizeAlloc()` allocates memory for the AutoGEM maximum element size mesh control data.

The function `ProMechmcautogemelemsizeSizeGet()` returns the maximum element size in the mesh. Use the function `ProMechmcautogemelemsizeSizeSet()` to set the element size.

Use the function `ProMechmcautogemelemsizeFree()` to free the memory containing the maximum element size mesh control data structure.

Accessing Edge Distribution Mesh Control Data

The functions described in this section provide access to the edge distribution mesh control data.

Functions Introduced:

- **ProMechmcedgedistrAlloc()**
- **ProMechmcedgedistrNodesGet()**
- **ProMechmcedgedistrNodesSet()**
- **ProMechmcedgedistrRatioGet()**
- **ProMechmcedgedistrRatioSet()**
- **ProMechmcedgedistrStrictGet()**
- **ProMechmcedgedistrStrictSet()**
- **ProMechmcedgedistrFree()**

The function `ProMechmcedgedistrNodesGet()` returns the minimum number of nodes that are distributed along the selected edge or curve. Use the function `ProMechmcedgedistrNodesSet()` to set the number of nodes for the mesh control data.

The function `ProMechmcedgedistrRatioGet()` returns the ratio of the first interval on the edge or curve to the last interval on the edge or curve. Use the function `ProMechmcedgedistrRatioSet()` to set the ratio for the mesh control data.

The function `ProMechmcedgedistrStrictGet()` returns a boolean flag indicating whether the number of nodes must be used exactly. Use the function `ProMechmcedgedistrStrictSet()` to set the maximum allowable nodes for the mesh control data.

Accessing AutoGEM Isolation Data

The functions in this section provide access to the entities that AutoGEM can detect and isolate using mesh refinement.

Functions Introduced:

- **ProMechmcautogemisolateexcludeAlloc()**
- **ProMechmcautogemisolateexcludeExcludeGet()**
- **ProMechmcautogemisolateexcludeExcludeSet()**
- **ProMechmcautogemisolateexcludeFree()**
- **ProMechmcautogemisolateexcludeShellIsolMaxSizeGet()**
- **ProMechmcautogemisolateexcludeShellIsolTypeGet()**
- **ProMechmcautogemisolateexcludeSolidIsolMaxSizeGet()**
- **ProMechmcautogemisolateexcludeSolidIsolTypeGet()**
- **ProMechmcautogemisolateexcludeShellIsolMaxSizeSet()**
- **ProMechmcautogemisolateexcludeShellIsolTypeSet()**
- **ProMechmcautogemisolateexcludeSolidIsolMaxSizeSet()**
- **ProMechmcautogemisolateexcludeSolidIsolTypeSet()**

The function `ProMechmcautogemisolateexcludeExcludeGet()` returns the value of the exclude boolean for the AutoGEM isolation for exclusion mesh control data. Use the function `ProMechmcautogemisolateexcludeExcludeSet()` to set the value of the exclude boolean for the AutoGEM isolation for exclusion mesh control data.

The function `ProMechmcautogemisolateexcludeShellIsolMaxSizeGet()` returns the value of maximum element size of isolation of shells for the AutoGEM isolation for exclusion mesh control data. Use the function `ProMechmcautogemisolateexcludeShellIsolMaxSizeSet()` to set the value of maximum element size of isolation of shells for the AutoGEM isolation.

The function `ProMechmcautogemisolateexcludeShellIsolTypeGet()` returns the value of shell isolation type for the AutoGEM isolation for exclusion mesh control data. Use the function `ProMechmcautogemisolateexcludeShellIsolTypeSet()` to set the value of shell isolation type for the AutoGEM isolation for exclusion mesh control data.

The function `ProMechmcautogemisolateexcludeSolidIsolMaxSizeGet()` returns the value of maximum element size of isolation of solids for the AutoGEM isolation for exclusion mesh control data. Use the function `ProMechmcautogemisolateexcludeSolidIsolMaxSizeSet()` to set the maximum element size of isolation of solids for the AutoGEM isolation.

The function `ProMechmcautogemisolateexcludeSolidIsolTypeGet()` returns the value of solid isolation type for the AutoGEM isolation for exclusion mesh control data. Use the function `ProMechmcautogemisolateexcludeSolidIsolTypeSet()` to set the value of the solid isolation type for the AutoGEM isolation.

Accessing the Displacement Coordinate System Data

The functions described in this section provide read and write access to the displacement coordinate system control data.

Functions Introduced:

- **ProMechmcdisplacementcsysAlloc()**
- **ProMechmcdisplacementcsysCsysGet()**
- **ProMechmcdisplacementcsysCsysSet()**
- **ProMechmcdisplacementcsysFree()**

The function `ProMechmcdisplacementcsysCsysGet()` returns the coordinate system used to display the results for the nodes associated with points, edges, curves, or surfaces.

The function `ProMechmcdisplacementcsysCsysSet()` sets the co-ordinate system for the mesh control data.

Accessing the Mesh Control Element Size Data

Functions Introduced:

- **ProMechmcelemsizeSizeGet()**
- **ProMechmcelemsizeSizeSet()**
- **ProMechmcelemsizeFree()**

The function `ProMechmcelemsizeSizeGet()` returns the size of the elements for the mesh control data. Use the function `ProMechmcelemsizeSizeSet()` to set the size of the elements for the mesh control data.

Accessing the Mesh Control Shell Coordinate System Data

Functions Introduced:

- **ProMechmcshellcsysCsysGet()**
- **ProMechmcshellcsysCsysSet()**
- **ProMechmcshellcsysDirectionGet()**
- **ProMechmcshellcsysDirectionSet()**
- **ProMechmcshellcsysFree()**

The function `ProMechmcshellcsysCsysGet()` returns the coordinate system specified for the mesh control data. Use the function `ProMechmcshellcsysCsysSet()` to set the coordinate system specified for the mesh control data.

The function `ProMechmcshellcsysDirectionGet()` returns the positive direction along the x, y, and z axis. Use the function `ProMechmcshellcsysDirectionSet()` to set the positive direction along the x, y, and z axis.

Accessing the Mesh Control Hard Point Data

Functions Introduced:

- **ProMechmchardpntNodeGet()**
- **ProMechmchardpntNodeSet()**
- **ProMechmchardpntFree()**

The function `ProMechmchardpntNodeGet()` returns the id of the node that is defined as a hard point. Use the function `ProMechmchardpntNodeSet()` to set the id of the node for the mesh control data.

Accessing the Mesh Control ID Offset Data

Functions Introduced:

- **ProMechmcidsoffsetOffsetGet()**
- **ProMechmcidsoffsetOffsetSet()**
- **ProMechmcidsoffsetFree()**

The function `ProMechmcidoffsetOffsetGet()` returns a positive integer value to be added to the IDs of each node, element, and local mesh entity. Use the function `ProMechmcidsoffsetOffsetSet()` to set the offset value for the mesh control data.

Accessing the Mesh Control Numbering Data

Functions Introduced:

- **ProMechmcnumberingFirstGet()**
- **ProMechmcnumberingFirstSet()**
- **ProMechmcnumberingIncrementGet()**
- **ProMechmcnumberingIncrementSet()**
- **ProMechmcnumberingLastGet()**
- **ProMechmcnumberingLastSet()**
- **ProMechmcnumberingFree()**

The function `ProMechmcnumberingFirstGet()` returns the first ID for the nodes, elements, and local mesh entities. Use the function `ProMechmcnumberingFirstSet()` to set the first value of the mesh control data.

The function `ProMechmcnumberingIncrementGet()` returns the increment id for the nodes, elements, and local mesh entities. Use the function `ProMechmcnumberingIncrementSet()` to set the increment value for the mesh control data.

The function `ProMechmcnumberingLastGet()` returns the last ID for the nodes, elements, and local mesh entities. Use the function `ProMechmcnumberingLastSet()` to set the last value for the mesh control data.

Accessing the Suppressed Mesh Control Data

Functions Introduced:

- **ProMechmcsuppressTypeGet()**
- **ProMechmcsuppressTypeSet()**
- **ProMechmcsuppressFree()**

The function `ProMechmcsuppressTypeGet()` returns the type of mesh control data that should be suppressed. Use the function `ProMechmcsuppressTypeSet()` to set the type of data to be suppressed.

Welds

Welds are used to bridge gaps that are formed during shell compression between plates that have been mated because they touch or overlap. Welds use the `ProType` field in the `ProMechitem` structure as `PRO_SIMULATION_WELD`.

Functions Introduced:

- **ProMechweldReferencesGet()**
- **ProMechweldReferencesSet()**
- **ProMechweldTypeGet()**
- **ProMechweldperimeterAlloc()**
- **ProMechweldPerimeterdataGet()**
- **ProMechweldPerimeterdataSet()**
- **ProMechweldperimeterEdgesGet()**
- **ProMechweldperimeterEdgesSet()**
- **ProMechweldperimeterFree()**
- **ProMechweldedgeAlloc()**
- **ProMechweldedgeEdgeGet()**
- **ProMechweldedgeEdgeSet()**
- **ProMechweldedgeThicknessGet()**
- **ProMechweldedgeThicknessSet()**
- **ProMechweldedgeMaterialidGet()**
- **ProMechweldedgeMaterialidSet()**
- **ProMechweldedgeFree()**
- **ProMechweldedgeProarrayFree()**
- **ProMechweldspotAlloc()**
- **ProMechweldSpotdataGet()**
- **ProMechweldSpotdataSet()**
- **ProMechweldspotPointsGet()**
- **ProMechweldspotPntsSet()**
- **ProMechweldspotDiameterGet()**
- **ProMechweldspotDiameterSet()**

- **ProMechweldspotMaterialIdGet()**
- **ProMechweldspotMaterialIdSet()**
- **ProMechweldspotFree()**
- **ProMechweldendSet()**
- **ProMechweldEnddataGet()**
- **ProMechweldEnddataSet()**
- **ProMechweldendAlloc()**
- **ProMechweldendTypeGet()**
- **ProMechweldendTypeSet()**
- **ProMechweldendExtendAdjacentSurfacesGet()**
- **ProMechweldendExtendAdjacentSurfacesSet()**
- **ProMechweldendFree()**
- **ProMechweldFeaturedataGet()**
- **ProMechweldFeaturedataSet()**
- **ProMechweldfeatureAlloc()**
- **ProMechweldfeatureMaterialidGet()**
- **ProMechweldfeatureOverrideflagGet()**
- **ProMechweldfeatureThicknessGet()**
- **ProMechweldedgeMaterialidSet()**
- **ProMechweldfeatureOverrideflagSet()**
- **ProMechweldfeatureThicknessSet()**
- **ProMechweldfeatureFree()**

The function `ProMechweldReferencesGet()` returns the geometric entities selected to create the weld. Use the function `ProMechweldReferencesSet()` to set the references for the specified weld.

The function `ProMechweldTypeGet()` returns the type of weld used to connect the gaps. The types of weld are:

- `PRO_MECH_WELD_PERIMETER`—Specifies a perimeter weld. Perimeter welds are used to connect parallel plates along the perimeter of one of the plates in an assembly model.
- `PRO_MECH_WELD_END`—Specifies an end weld. End welds are used to connect plates in assembly models.

- PRO_MECH_WELD_SPOT—Specifies a spot weld. Spot welds are used to connect two parallel surfaces at the specified datum point.
- PRO_MECH_WELD_FEAT—Specifies a Weld Feature. This weld connection can be used to select Fillet and Groove types of weld connections for inclusion in a mid-surface compressed model.

The function ProMechweldPerimeterdataGet() provides access to the perimeter weld data structure. Use the function ProMechweldPerimeterdataSet() to set the perimeter weld data structure.

Use the function ProMechweldperimeterFree() to free the memory contained in the perimeter weld data structure.

The function ProMechweldperimeterEdgesGet() returns the perimeter edges of the top plate to be connected to the base plate. Use the function ProMechweldperimeterEdgesSet() to set the perimeter edges in the perimeter weld data.

The function ProMechweldedgeEdgeGet() returns the id of the edge selected to create the perimeter weld. Use the function ProMechweldedgeEdgeSet() to set the id of the edge.

The function ProMechweldedgeThicknessGet() returns the thickness of the weld on the selected edge in the perimeter weld. Use the function ProMechweldedgeThicknessSet() to set the thickness of the weld edge data.

The function ProMechweldedgeMaterialidGet() returns the type of material used in shell elements created on the Perimeter Weld surfaces.

Use the function ProMechweldedgeMaterialidSet() to select the material for the shell elements created on the Perimeter Weld surfaces. Specify the material ID as the input parameter of this function. In addition to the materials already present in the model, the input can be the material ID of one of the following surfaces:

- PRO_MECH_WELD_MTL_BASE—Specifies the base surface to which the weld extends.
- PRO_MECH_WELD_MTL_DOUBLER—Specifies the doubler surface on which the Perimeter weld is placed.

The functions ProMechweldedgeFree() and ProMechweldedgeProarrayFree() free the memory containing the edge data structure.

The function ProMechweldperimeterFree() frees the memory containing the data structure for the perimeter weld.

The function `ProMechweldSpotdataGet()` provides access to the spot weld data structure. Use the function `ProMechweldSpotdataSet()` to assign data to the spot weld data structure.

The function `ProMechweldspotPointsGet()` returns the datum points at which the spot weld is located. Use the function `ProMechweldspotPntsSet()` to set the points for the spot weld data.

The function `ProMechweldspotDiameterGet()` returns the diameter for the spot weld. Use the function `ProMechweldspotDiameterSet()` to set the diameter for the spot weld data.

The function `ProMechweldspotMaterialIdGet()` returns the material for the spot weld. Use the function `ProMechweldspotMaterialIdSet()` to set the material for the spot weld.

Use the function `ProMechweldspotFree()` to free the memory contained in the spot weld data structure.

Use the function `ProMechweldendSet()` to set the weld to be an end weld.

The function `ProMechweldEnddataGet()` returns the end weld data structure.

Use the function `ProMechweldendAlloc()` to allocate memory for the end weld data handle.

Use the method `ProMechweldEnddataSet()` to set the end weld data structure.

The function `ProMechweldendTypeGet()` returns the type of end weld defined between different surfaces. The valid types of end welds are:

- `PRO_MECH_WELD_END_SINGLE_TO_SINGLE`—Specifies a weld defined between a side surface and a shell or shell paired surface.
- `PRO_MECH_WELD_END_MANY_TO_SINGLE`—Specifies a weld defined from a solid surface to a solid surface.
- `PRO_MECH_WELD_END_SINGLE_TO_MANY`—Specifies a weld surface defined from a solid surface to a shell surface.

Use the function `ProMechweldendTypeSet()` to set the type of end welds.

The function `ProMechweldendExtendAdjacentSurfacesGet()` specifies whether or not surfaces adjacent to the selected source surface will be extended to the target surface to create the weld. Use the function `ProMechweldendExtendAdjacentSurfacesSet()` to control the placement of the weld on the extension of the adjacent surfaces.

Use the function `ProMechweldendFree()` to free the memory of the end weld data structure.

The function `ProMechweldFeaturedataGet()` returns the Weld Feature data structure.

Use the function `ProMechweldfeatureAlloc()` to allocate memory for the Weld Feature data structure.

Use the function `ProMechweldFeaturedataSet()` to set the Weld Feature data structure.

The function `ProMechweldfeatureOverrideflagGet()` specifies if the material and thickness properties that will be applied to the shell idealizations resulting from the weld definitions are inherited from the Weld Feature or not.

Specify the input value of the function `ProMechweldfeatureOverrideflagSet()` as `true` to specify the properties for the shell idealizations. Specify the value as `false` to inherit the properties from the Weld Feature.

You can use the following functions to access and modify the material property and thickness of the Weld Feature only if you specify the input value as `true` for the function `ProMechweldfeatureOverrideflagSet()` or you select the option **Override Weld Feature Settings** from the Creo Simulate user interface.

The function `ProMechweldfeatureMaterialidGet()` returns the material type used for the Weld Feature. Use the function `ProMechweldfeatureMaterialidSet()` to set the material for the shell idealizations. Specify the material ID as the input parameter of this function.

The function `ProMechweldfeatureThicknessGet()` returns the thickness of the Weld Feature. Use the function `ProMechweldfeatureThicknessSet()` to set the thickness of the shell idealizations.

Use the function `ProMechweldfeatureFree()` to free the memory contained in the Weld Feature data structure.

Creo Simulate Features

Functions Introduced:

- **ProMechFeaturePromote()**

The function `ProMechFeaturePromote()` promotes the specified Creo Simulate feature making it accessible in Creo Parametric. The promoted feature however cannot be transferred back to Creo Simulate .

Validating New and Modified Simulation Objects

You can check the validity of simulation objects that are created new or modified. During validation, an error object is created or updated with error, warning, and information messages.

Functions Introduced:

- **ProMechitemValidate()**
- **ProMecherrobjDataGet()**
- **ProMecherrobjMessageGet()**

The function `ProMechitemValidate()` validates the specified Creo Simulate item and returns an error object that collects the error, warning, and information messages. The function sets the error checking mode, validates the Creo Simulate item, resets the error checking mode, and returns the error object.

The error, warning and information messages are given by the enumerated type `ProMechErrobjType`, which has the following values:

- `PRO_MECH_ERROBJ_ERROR`
- `PRO_MECH_ERROBJ_WARNING`
- `PRO_MECH_ERROBJ_INFO`

The function `ProMecherrobjDataGet()` returns the number of events of an error object for the given `ProMechErrobjType`. To get the event message at a given index in an error object, you can use the function `ProMecherrobjMessageGet()`.

70

Creo Simulate: Geometry

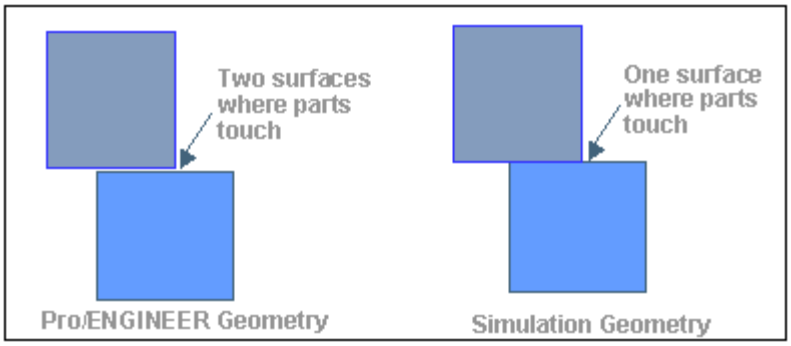
In Creo Parametric, when analysis is performed on a model (in either FEM or Native), the geometry that is seen on the screen is processed to create the Creo Simulate Geometry. This special temporary geometry is more suitable for analysis than the standard Creo Parametric geometry.

Introduction

Creo Simulate Geometry differs from that of standard Creo Parametric geometry in several ways.

- Creo Simulate geometry provides a non-manifold representation of assembly models.

Non-manifold geometry

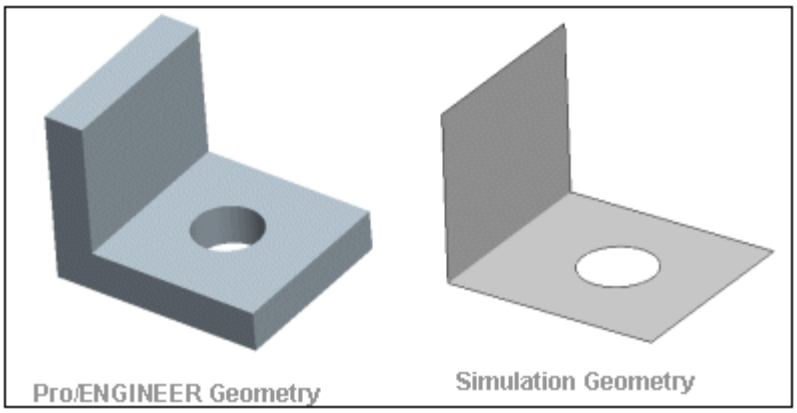


In assemblies, when two parts are mated, there exist two surfaces where the parts are mated, each of which does not know the existence of the other.

For analysis applications (particularly in meshing), you want the geometry to have only one surface in this mated area

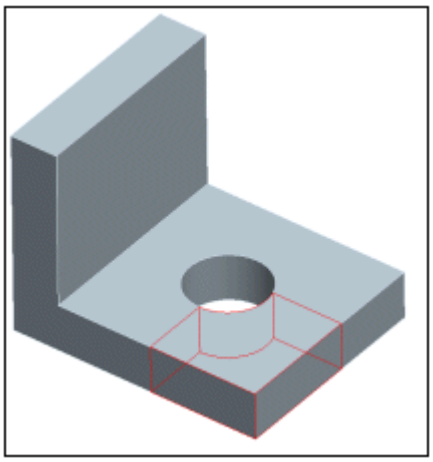
- Creo Simulate geometry provides mid-surface geometry for models which have shells defined.

Mid-surface geometry



- Creo Simulate geometry provides volume and surface region information.

“Tagged” region



Without access to the Creo Simulate geometry, it is difficult to use the information on volume and surface region.

The Creo Simulate geometry identifies which surfaces are "duplicated" at the boundaries within the solid. This makes it easy for external applications to re-create the non-manifold solid.

- Creo Simulate Geometry model is devoid of the undesirable features (from the point of view of analysis) present in the Creo Parametric model such as
  - Small, localized misalignments in the geometry
  - Cusps
  - Very small sliver surfaces

Obtaining Creo Simulate Geometry from Creo Parametric TOOLKIT

Creo Parametric represents the top level access to Creo Simulate geometry as an opaque handle called `ProMechModel`. The `ProMechModel` handle is generated upon request and can be accessed until it is freed by the application.

There are several modes which can be used to generate the model and its geometry:

- `PRO_MECH_MODEL_SOLID` signifies solid surfaces only. Shells will be ignored.
- `PRO_MECH_MODEL_SHELL` signifies shell-compressed surfaces. Non shell-compressed surfaces will not be included.
- `PRO_MECH_MODEL_MIXED` signifies both solid and shell-compressed surfaces.

- `PRO_MECH_MODEL_BOUNDARY` signifies shell surfaces occurring on the solid boundary.
- `PRO_MECH_MODEL_QUILT` signifies shell surfaces on quilts. This option requires that quilts be specified as additional entities to be processed.
- `PRO_MECH_MODEL_BAR` signifies bars on given datum curves. This option requires that curves be specified as additional entities to be processed.

It is not always necessary to have a Creo Simulate license to generate the Creo Simulate geometry. However, some situations do require the license:

- Use of the options `PRO_MECH_MODEL_SHELL`, `PRO_MECH_MODEL_MIXED` and `PRO_MECH_MODEL_BOUNDARY` is not permitted unless Creo Simulate is active. Please see the description for `ProMechanicaIsActive()` in the chapter Accessing Creo Simulate Items on page 1856.
- If any component of the model contains surface or volume region feature, Creo Simulate must be active to access the geometry of these regions.

Functions Introduced:

- **ProMechmodeldataAlloc()**
- **ProMechmodeldataTypeSet()**
- **ProMechmodeldataEntitiesSet()**
- **ProMechmodeldataPreserveidsSet()**
- **ProMechmodeldataMergeCoincSolidEdgesSet()**
- **ProMechmodelGet()**
- **ProMechmodelFree()**
- **ProMechmodeldataFree()**

Use the function `ProMechmodeldataAlloc()` to allocate an input data structure for generation of Creo Simulate geometry.

Use the function `ProMechmodeldataTypeSet()` to assign the type of model to be generated.

Use the function `ProMechmodeldataEntitiesSet()` to assign the additional datum points, quilts, and curves to be included in the geometric processing.

Use the function `ProMechmodeldataPreserveidsSet()` to set the preservation of IDs. In the event of the "Preserve IDs" flag being set, Creo Simulate tries to preserve the IDs of surfaces and edges in Creo Simulate geometry between different calls to `ProMechmodelGet()`. This happens even if the original geometry of the model has changed slightly, but the success of this attempt to preserve geometry IDs is not always guaranteed.

Use the function `ProMechmodeldataMergeCoincSolidEdgesSet()` to set the flag that determines whether coincident solid edges will be merged. By default, the flag is `PRO_B_FALSE`, that is, the solid edges will not be merged.

Use the function `ProMechmodelGet()` to obtain the root handle of a Creo Simulate geometry model. The input arguments of this function are:

- *solid*—signifies the root solid model (part or assembly).
- *data*—signifies the options for the generation of geometry.

The output arguments are as follows:

- *mech_model* signifies the root handle of the generated geometry.
- *status* signifies the status of generation and can have the following values:
  - `PRO_MECH_MODEL_SUCCESS` signifies success.
  - `PRO_MECH_MODEL_GENERAL_FAILURE` signifies general failure.
  - `PRO_MECH_MODEL_USER_INTERRUPT` signifies that the user interrupted the process before it completed.
  - `PRO_MECH_MODEL_SHELL_NO_PAIRS` signifies that no shell pairs were defined (for options that used shell compression).
  - `PRO_MECH_MODEL_SHELL_SOME_PAIRS` signifies that some paired and some unpaired surfaces exist.(for options that used shell compression).

Use the function `ProMechmodeldataFree()` to free a data handle used for generation of a Creo Simulate geometry model. Use the function `ProMechmodelFree()` to free a Creo Simulate model handle. This invalidates all geometric entities obtained from this handle.

Accessing the ProMechModel

Functions Introduced:

- **ProMechmodelMdlGet()**
- **ProMechmodelSolidVisit()**
- **ProMechmodelPointVisit()**
- **ProMechmodelCurveVisit()**
- **ProMechmodelToleranceGet()**

The function `ProMechmodelMdlGet()` can be used to obtain the root solid model used for generation of this Creo Simulate model.

The function `ProMechmodelSolidVisit()` visits the solid volumes that make up a Creo Simulate geometry model.

The function `ProMechmodelPointVisit()` visits the datum points that are included in a Creo Simulate geometry model.

Note

only datum points that have Creo Simulate items referencing them will be included unless you pass the additional points as input from `ProMechmodeldataEntitiesSet()`.

The function `ProMechmodelCurveVisit()` visits the composite curves that are included in a Creo Simulate geometry model.

Note

only datum points that have Creo Simulate items referencing them will be included unless you pass the additional points as input from `ProMechmodeldataEntitiesSet()`.

Use the function `ProMechmodelToleranceGet()` to obtain the overall tolerance or epsilon value used for the preparation of the Creo Simulate geometry.

Accessing the ProMechSolid

An opaque handle called a `ProMechSolid` represents a solid volume member of the model (typically, an assembly member).

Functions Introduced:

- **ProMechsolidVisitAction()**
- **ProMechsolidFilterAction()**
- **ProMechsolidIdGet()**
- **ProMechsolidModelGet()**
- **ProMechsolidTypeGet()**
- **ProMechsolidAsmcomppathGet()**
- **ProMechsolidSurfaceVisit()**
- **ProMechsolidEdgeVisit()**
- **ProMechsolidVertexVisit()**

The function types `ProMechsolidVisitAction()` and `ProMechsolidFilterAction()` are used as arguments to functions that visit `ProMechSolid` objects.

Use the function `ProMechsolidIdGet()` to obtain the ID of the solid volume. This ID is an index in the array of generated solid volumes and is not persistent among different calls to `ProMechmodelCreate()`.

Use the function `ProMechsolidModelGet()` to obtain the root Creo Simulate geometry model for this solid model.

Use the function `ProMechsolidTypeGet()` to obtain the solid volume type.

Use the function `ProMechsolidAsmcomppathGet()` to obtain the assembly component path from the top level assembly to this solid.

Use the function `ProMechsolidSurfaceVisit()` to visit the surfaces that are included in a given solid volume.

Use the function `ProMechsolidEdgeVisit()` to visit the edges that are included in a given solid volume.

Use the function `ProMechsolidVertexVisit()` to visit the vertices that are included in a given solid volume.

Accessing Creo Simulate ProMechSurface

An opaque handle called a `ProMechSurface` represents a Creo Simulate geometry surface. A surface can be of the following types:

- `PRO_MECH_FACE_SOLID`—A solid surface
- `PRO_MECH_FACE_SHELL`—A shell (mid-plane) surface
- `PRO_MECH_FACE_SIDE`—A shell side surface
- `PRO_MECH_FACE_SHELL_QLT`—A quilt surface
- `PRO_MECH_FACE_BOUNDARY`—A solid boundary surface
- `PRO_MECH_FACE_PERIM_WELD`—A generated surface for a perimeter weld object

Functions Introduced:

- **ProMechsurfaceVisitAction()**
- **ProMechsurfaceFilterAction()**
- **ProMechsurfaceIdGet()**
- **ProMechsurfaceOwnerGet()**
- **ProMechsurfaceTypeGet()**
- **ProMechsurfaceContourVisit()**
- **ProMechsurfaceEdgeVisit()**
- **ProMechsurfaceVertexVisit()**

- **ProMechsurfaceIncontactfacesGet()**
- **ProMechsurfaceAncestorsGet()**

The function types `ProMechsurfaceVisitAction()` and `ProMechsurfaceFilterAction()` are used as arguments to functions that visit `ProMechSurface` objects.

Use the function `ProMechsurfaceIdGet()` to obtain the surface ID of the surface.

Note

This ID is not persistent and is not related to the Creo Parametric surface ID.

Use the function `ProMechsurfaceOwnerGet()` to obtain the owner `ProMechsolid` of the surface.

Use the function `ProMechsurfaceTypeGet()` to obtain the type of surface.

Use the function `ProMechsurfaceContourVisit()` to visit the contours that are included in a given surface.

Use the function `ProMechsurfaceEdgeVisit()` to visit the edges that are included in a given surface.

Use the function `ProMechsurfaceVertexVisit()` to visit vertices that are included in a given surface.

Use the function `ProMechsurfaceAncestorsGet()` to obtain the ancestor surfaces for the given surface. These are the actual Creo Parametric geometry surfaces used to construct this Creo Simulate surface.

Geometry Evaluation of ProMechSurface

For information about how Creo Parametric TOOLKIT represents surface geometry, see the chapter on Core: 3D Geometry on page 170 and the appendix, Element Trees: References on page 799.

Functions Introduced:

- **ProMechsurfaceParamEval()**
- **ProMechsurfaceUvpntVerify()**
- **ProMechsurfaceTessellationGet()**
- **ProMechsurfaceDataGet()**
- **ProMechsurfaceToNURBS()**

- **ProMechsurfaceTransformGet()**
- **ProMechsurfaceThicknessEval()**

Use the function `ProMechsurfaceParamEval()` to find the corresponding UV point on the Creo Simulate geometry surface on the basis of the XYZ point.

Use the function `ProMechsurfaceUvpntVerify()` to verify whether the specified UV point lies within the boundaries of the Creo Simulate surface.

Use the function `ProMechsurfaceTessellationGet()` to calculate the tessellation for the provided Creo Simulate surface.

Use the function `ProMechsurfaceDataGet()` to obtain the geometric representation of the surface.

Use the function `ProMechsurfaceToNURBS()` to obtain the NURBS representation of the Creo Simulate surface.

Use the function `ProMechsurfaceTransformGet()` to obtain UV transform between the two surfaces in contact.

Use the function `ProMechsurfaceThicknessEval()` to obtain the thickness of the shell Creo Simulate surface at the given UV point.

Accessing ProMechContour

An opaque handle called a `ProMechContour` represents a contour member of the model. See the chapter on Core: 3D Geometry on page 170 for a discussion of contours.

Functions Introduced:

- **ProMechcontourVisitAction()**
- **ProMechcontourFilterAction()**
- **ProMechcontourIdGet()**
- **ProMechcontourSurfaceGet()**
- **ProMechcontourTraversalGet()**
- **ProMechcontourEdgeVisit()**
- **ProMechcontourUvpntVerify()**
- **ProMechcontourAreaEval()**
- **ProMechcontourContainingContourGet()**

The function types `ProMechcontourVisitAction()` and `ProMechcontourFilterAction()` are used as arguments to functions that visit `ProMechContour` objects.

Use the function `ProMechcontourIdGet()` to obtain the ID for a given contour. This ID is unique within the surface that owns the contour.

Use the function `ProMechcontourSurfaceGet()` to obtain the surface that contains the contour.

Use the function `ProMechcontourTraversalGet()` to obtain the contour traversal.

Use the function `ProMechcontourEdgeVisit()` to visit the edges that make up a contour.

Use the function `ProMechcontourUvpntVerify()` to verify whether the specified UV point lies within the given Creo Simulate contour.

Use the function `ProMechcontourAreaEval()` to find the surface area inside the given outer contour, accounting for internal voids.

The function `ProMechcontourContainingContourGet()` returns the containing contour for a Creo Simulate contour object.

Accessing ProMechEdge

An opaque handle called a `ProMechEdge` represents an edge member of the model.

Functions Introduced:

- **ProMechedgeVisitAction()**
- **ProMechedgeFilterAction()**
- **ProMechedgeIdGet()**
- **ProMechedgeOwnerGet()**
- **ProMechedgeSurfaceVisit()**
- **ProMechedgeContourVisit()**
- **ProMechedgeEndpointsGet()**
- **ProMechedgeIncontactedgesGet()**
- **ProMechedgeAncestorsGet()**

The function types `ProMechedgeVisitAction()` and `ProMechedgeFilterAction()` are used as arguments to functions that visit `ProMechEdge` objects.

Use the function `ProMechedgeIdGet()` to obtain the ID of the given edge.

Note

This ID is not persistent and is not related to the Creo Parametric edge ID.

Use the function `ProMechedgeOwnerGet()` to obtain the owner `ProMechSolid` of the given edge.

Use the function `ProMechedgeSurfaceVisit()` to visit the surfaces that share this edge.

Use the function `ProMechedgeContourVisit()` to visit the contours that contain this edge.

Use the function `ProMechedgeEndpointsGet()` to obtain the endpoints of the given edge.

Use the function `ProMechedgeIncontactedgesGet()` to obtain the list of edges that are in contact with the given edge.

Use the function `ProMechedgeAncestorsGet()` to obtain the ancestor edges for the given edge. These are the actual Creo Parametric geometry edges used to construct this Creo Simulate edge.

Geometry Evaluation of ProMechEdge

For information about how Creo Parametric TOOLKIT represents edge geometry, see the chapter on Core: 3D Geometry on page 170 and the appendix, Element Trees: References on page 799.

Functions Introduced:

- **ProMechedgeUvdataEval()**
- **ProMechedgeXyzdataEval()**
- **ProMechedgeParamEval()**
- **ProMechedgeLengthEval()**
- **ProMechedgeLengthT1T2Eval()**
- **ProMechedgeParamByLengthEval()**
- **ProMechedgeTessellationGet()**
- **ProMechedgeDataGet()**
- **ProMechedgeToNURBS()**
- **ProMechedgeDirectionGet()**
- **ProMechedgeReldirGet()**

Use the function `ProMechedgeUvdataEval()` to evaluate the Creo Simulate edge in the UV space of the given surface.

Use the function `ProMechedgeXyzdataEval()` to evaluate the Creo Simulate edge parameter point in XYZ space.

Use the function `ProMechedgeParamEval()` to find the corresponding normalized parameter on the Creo Simulate edge by XYZ point.

Use the function `ProMechedgeLengthEval()` to obtain the length of the edge.

Use the function `ProMechedgeLengthT1T2Eval()` to find the length of the Creo Simulate edge between the given parameters.

Use the function `ProMechedgeParamByLengthEval()` to find the parameter of the point located at the given length from the given parameter.

Use the function `ProMechedgeTessellationGet()` to get the edge tessellation for the Creo Simulate edges.

Use the function `ProMechedgeToNURBS()` to obtain the NURBs representation of the Creo Simulate edge.

Use the function `ProMechedgeReldirGet()` to obtain the relative direction of two Creo Simulate edges in contact.

Use the function `ProMechedgeDataGet()` to obtain the geometric representation of the edge.

Use the function `ProMechedgeDirectionGet()` to obtain the edge direction with respect to the given contour.

Accessing ProMechVertex

An opaque handle called a `ProMechVertex` represents a vertex member of the model.

Functions Introduced:

- **ProMechvertexVisitAction()**
- **ProMechvertexFilterAction()**
- **ProMechvertexIdGet()**
- **ProMechvertexOwnerGet()**
- **ProMechvertexPointGet()**
- **ProMechvertexSurfaceVisit()**
- **ProMechvertexEdgeVisit()**
- **ProMechvertexIncontactverticesGet()**

The function types `ProMechvertexVisitAction()` and `ProMechvertexFilterAction()` are used as arguments to functions that visit `ProMechVertex` objects.

Use the function `ProMechvertexIdGet()` to obtain the ID of the given vertex.

Use the function `ProMechvertexOwnerGet()` to obtain the volume that owns this vertex.

Use the function `ProMechvertexPointGet()` to obtain the coordinate point for a given vertex.

Use the function `ProMechvertexSurfaceVisit()` to visit the surfaces that include this vertex.

Use the function `ProMechvertexEdgeVisit()` to visit the edges that contain this vertex.

Use the function `ProMechvertexIncontactverticesGet()` to obtain the list of vertices that are in contact with the given vertex.

Accessing ProMechPoint

An opaque handle called a `ProMechPoint` represents a datum point member of the model. By default, the Creo Simulate geometry will include only those points which have Creo Simulate loads or other items referencing them; you can generate additional points by including them as inputs to `ProMechmodeldataEntitiesSet()`.

Functions Introduced:

- **ProMechpointVisitAction()**
- **ProMechpointFilterAction()**
- **ProMechpointIdGet()**
- **ProMechpointOwnerGet()**
- **ProMechpointPointGet()**
- **ProMechpointPlacementtypeGet()**
- **ProMechpointPlacementsurfaceGet()**
- **ProMechpointPlacementedgeGet()**
- **ProMechpointPlacementvertexGet()**
- **ProMechpointAncestorsGet()**

The function types `ProMechpointVisitAction()` and `ProMechpointFilterAction()` are used as arguments to functions that visit `ProMechPoint` objects.

Use the function `ProMechpointIdGet()` to obtain the ID of the point.

Note

This ID is not persistent and is not related to the Creo Parametric point ID.

Use the function `ProMechpointOwnerGet()` to obtain the owner model of the given point.

Use the function `ProMechpointPointGet()` to obtain the coordinates of the point.

Use the function `ProMechpointPlacementtypeGet()` to obtain the placement type for the point. Following are list of possible placement types:

- `PRO_MECH_PNT_FREE` signifies that the point is not attached to a solid or shell.
- `PRO_MECH_PNT_FACE` signifies that the point lies on a `ProMechSurface`.
- `PRO_MECH_PNT_EDGE` signifies that the point lies on a `ProMechEdge`.
- `PRO_MECH_PNT_VERTEX` signifies that the point lies on a `ProMechVertex`.

Use the function `ProMechpointPlacementsurfaceGet()` to obtain the placement surface, if the placement type is `PRO_MECH_POINT_FACE`.

Use the function `ProMechpointPlacementedgeGet()` to obtain the placement edge, if placement type is `PRO_MECH_POINT_EDGE`.

Use the function `ProMechpointPlacementvertexGet()` to obtain the placement vertex, if the placement type is `PRO_MECH_POINT_VERTEX`.

Use the function `ProMechpointAncestorsGet()` to obtain the ancestor points for the given point. These are the actual Creo Parametric geometry points used to construct this Creo Simulate point.

Accessing ProMechCompositeCurve

An opaque handle called a `ProMechCompositeCurve` represents a composite curve member of the model. By default, the Creo Simulate geometry will include only those curves which have Creo Simulate loads or other items referencing them. You can generate additional curves by including them as inputs to `ProMechmodeldataEntitiesSet()`.

Functions Introduced:

- **ProMechcompositecurveVisitAction()**
- **ProMechcompositecurveFilterAction()**
- **ProMechcompositecurveIdGet()**
- **ProMechcompositecurveOwnerGet()**
- **ProMechcompositecurveCurveVisit()**
- **ProMechcompositecurveAncestorsGet()**

The function types `ProMechcompositecurveVisitAction()` and `ProMechcompositecurveFilterAction()` are used as arguments to functions that visit `ProMechCompositeCurve` objects.

Use the function `ProMechcompositecurveIdGet()` to obtain the ID of the composite curve.

Note

This ID is not persistent and is not related to the Creo Parametric composite curve ID.

Use the function `ProMechcompositecurveOwnerGet()` to obtain the owner model of the composite curve.

Use the function `ProMechcompositecurveCurveVisit()` to visit the curves that make up this composite curve.

Use the function `ProMechcompositecurveAncestorsGet()` to obtain the ancestor curves for the given curve. These are the actual Creo Parametric geometry curves used to construct this Creo Simulate curve.

Accessing ProMechCurve

An opaque handle called a `ProMechCurve` represents a curve member of the model. A curve is always the child of a composite curve.

Functions Introduced:

- **ProMechcurveVisitAction()**
- **ProMechcurveFilterAction()**
- **ProMechcurveTypeGet()**
- **ProMechcurveEdgeGet()**
- **ProMechcurveParentGet()**
- **ProMechcurveParamGet()**
- **ProMechcurveEndpointsGet()**
- **ProMechcurveAncestorsGet()**

The function types `ProMechcurveVisitAction()` and `ProMechcurveFilterAction()` are used as arguments to functions that visit `ProMechCurve` objects.

Use the function `ProMechcurveTypeGet()` to obtain the curve's type. Curves may be of the following types:

- `PRO_MECH_SEGMENT_FREE`
- `PRO_MECH_SEGMENT_ON_EDGE`

Use the function `ProMechcurveEdgeGet()` to obtain the edge, if the curve type is `PRO_MECH_CURVE_EDGE`.

Use the function `ProMechcurveParentGet()` to obtain the parent composite curve for this curve.

Use the function `ProMechcurveParamGet()` to obtain the parameter along the parent composite curve at which this curve begins.

Use the function `ProMechcurveEndpointsGet()` to obtain the endpoints of the curve.

Use the function `ProMechcurveAncestorsGet()` to obtain the ancestor curves for the specified Creo Simulate curve. The ancestor curves are the actual Creo Parametric geometry curves used to construct the Creo Simulate curve.

Geometry Evaluation of ProMechCurves

For information about how Creo Parametric TOOLKIT represents curve geometry, see the chapter on Core: 3D Geometry on page 170 and the appendix, Element Trees: References on page 799.

Functions Introduced:

- **ProMechcurveXyzdataEval()**
- **ProMechcurveParamEval()**
- **ProMechcurveLengthEval()**
- **ProMechcurveLengthT1T2Eval()**
- **ProMechcurveParamByLengthEval()**
- **ProMechcurveDataGet()**
- **ProMechcurveToNURBS()**

Use the function `ProMechcurveXyzdataEval()` evaluates the Creo Simulate curve parameter point in the XYZ point.

Use the function `ProMechcurveParamEval()` to find the corresponding normalized parameter on the Creo Simulate curve by XYZ point.

Use the function `ProMechcurveLengthEval()` to obtain the length of the curve.

Use the function `ProMechcurveLengthT1T2Eval()` to find the length of the Creo Simulate curve between the given parameters.

Use the function `ProMechcurveParamByLengthEval()` to find the parameter of the point located at the given length from the given parameter.

Use the function `ProMechcurveDataGet()` to obtain the geometric representation of the edge.

Use the function `ProMechcurveToNURBS()` to obtain the NURBS representation of the Creo Simulate curve.

To Create a Surface Region Feature

This section describes the use the header file `ProSurfReg.h` to create a surface region feature programmatically. The chapter Element Trees: Principles of Feature Creation on page 764 provides the necessary background for creating features; we recommend you read that material first.

Note

The Surface Region feature is available and can be regenerated only in the Creo Simulate environment.

The following figure shows the element tree for the Surface Region feature.

```
PRO_E_FEATURE_TREE
   |
   |-- PRO_E_FEATURE_TYPE
   |
   |-- PRO_E_STD_FEATURE_NAME
   |
   |-- PRO_E_SURFREG_SPLITTING_OPTION
   |
   |-- PRO_E_STD_SECTION
   |
   |-- PRO_E_STD_CURVE_COLLECTION_APPL
   |
   |-- PRO_E_STD_SURF_COLLECTION_APPL
```

The Surface Region feature element tree contains no non-standard element types. The following table describes special information about the elements in this tree.

Element ID	Value
`PRO_E_FEATURE_TYPE`	`PRO_FEAT_SPLIT_SURF`
`PRO_E_STD_FEATURE_NAME`	Specifies the name of the Surface Region feature. The default value is “Surface Region”. This element is optional.
`PRO_E_SURFREG_SPLITTING_OPTION`	Specifies the method to define the surface contour. Valid values are: `PRO_SURFREG_SKETCH`—Split the surface using a sketch. `PRO_SURFREG_CHAIN`—Split the surface using a chain.
`PRO_E_STD_SECTION`	Specifies a 2D section or a sketched section. Refer to

Element ID	Value
	the section Creating Features Containing Sections on page 1006 for details on how to create features that contain sketched sections. Refer to the section Creating Section Models on page 988 for details on creating 2D sections.
PRO_E_STD_CURVE_COLLECTION_APPL	Specifies a collection of selected curves or edges or both to copy.
PRO_E_STD_SURF_COLLECTION_APPL	Specifies a collection of selected surfaces to copy.

71

Creo Simulate: Finite Element Modeling (FEM)

This chapter contains descriptions of the Creo Parametric TOOLKIT functions that support Creo Parametric Finite Element Modeling (FEM).

Overview

The Finite Element Modeling (FEM) functions in this chapter are designed to give you access to data generated by the Pro/MESH module of Creo Parametric. You can do the following:

- Export a Pro/MESH output file to disk.

Exporting an FEA Mesh

Function Introduced:

- **ProFemmeshExport()**

The function `ProFemmeshExport()` generates the Finite Element Mesh based on the given parameters, and exports it to the specified file.

The function uses the data structure `ProFemmeshData`, which is defined as follows:

```
typedef struct pro_femmesh_data
{
  ProFemmeshType          mesh_type;
  ProFemshellmeshType     shell_type;
  int                     num_quilts;
  ProFemquiltref        * pro_quilt_ref_arr;
  ProFemanalysisType      analysis;
  ProFemelemshapeType     elem_shape;
  ProFemsolverType        solver;
  ProFemcsysref           csys_ref;
  int                     num_aux_csys;
  ProFemcsysref         * aux_csys_ref_arr;
}ProFemmeshData;
```

The `pro_femmesh_data` fields are as follows:

- `mesh_type`—The mesh type. The possible values are as follows:
  - `PRO_FEM_SOLID_MESH`—Mesh solid parts using tetrahedral solid mesh elements.
  - `PRO_FEM_SHELL_MESH`—Shell mesh using triangular or quadrangular mesh elements. This type is designed for meshing surfaces.
  - `PRO_FEM_MIXED_MESH`—Mesh models with a mixture of shell and tetrahedral mesh elements.
  - `PRO_FEM_QUILT_MESH`—A mesh for any simple or advanced shell idealizations created for quilt surfaces.
  - `PRO_FEM_BOUNDARY_MESH`—A shell mesh of triangular or quadrilateral elements on the model's exterior surfaces.

- PRO_FEM_BAR_MESH—A bar mesh for one dimensional idealizations.

- shell_type—The type of shell element. This field is ignored for a solid mesh. The possible values are as follows:
  - PRO_FEM_TRIANGLE
  - PRO_FEM_QUADRANGLE
- num_quilts—The quilt identifier.
- pro_quilt_ref_arr—An array of references of quilt surfaces in the assembly.
- analysis—The analysis type. The possible values are as follows:
  - PRO_FEM_ANALYSIS_STRUCTURAL—Structural analysis, including stress, strain, thermal stress, and displacement.
  - PRO_FEM_ANALYSIS_MODAL—Modal analysis, including the constraint sets applied to the model.
  - PRO_FEM_ANALYSIS_THERMAL—Thermal analysis, including temperature, heat flux, and heat gradient.
- elem_shape—The type of element to be used for the solver.
  - PRO_FEM_MIDPNT_LINEAR—Linear elements are used for the analysis. This includes corner nodes, straight edges, and planar faces.
  - PRO_FEM_MIDPNT_PARABOLIC—Parabolic elements are used for the analysis.
  - PRO_FEM_MIDPNT_PARABOLIC_FIXED—The excessively curved edges of solid and shell mesh parabolic elements are slightly straightened and then used for analysis.

  The number of nodes that are as follows:

Tetrahedron	PRO_FEA_LINEAR: 4 nodes PRO_FEA_PARABOLIC: 10 nodes
Triangle	PRO_FEA_LINEAR: 3 nodes PRO_FEA_PARABOLIC: 6 nodes
Quadrangle	PRO_FEA_LINEAR: 4 nodes PRO_FEA_PARABOLIC: 8 nodes

- solver—The type of solver used for analysis. The possible values are:
  - PRO_FEM_FEAS_ANSYS—Specifies an ANSYS solver.
  - PRO_FEM_FEAS_NASTRAN—Specifies a NASTRAN solver.

    - `PRO_FEM_FEAS_NEUTRAL`—Specifies other solvers that support the FEM Neutral file format for analysis.
- `csys_ref`—An array of geometric references.
- `num_aux_csys`—Additional coordinate system to be included in the analysis.
- aux_csys_ref_arr—An array of geometric references for the auxiliary coordinate system.

> **Note**
>
> Prior to calling this function, the model (pro_solid) should be displayed in the graphics window.

The input arguments of this function are as follows:

- *pro_solid*— The handle of a Creo Parametric model (part or assembly).
- *p_mesh_data*—The pointer to the data structure containing the mesh generation parameters.
- *file_name*—The file name to export mesh to.

This function supersedes the function `pro_export_fea_mesh()`

72

Mechanism Design: Mechanism Features

This chapter describes the programmatic creation of mechanism modeling entities such as springs and dampers as Creo Parametric features.

The chapter also explains for to add motor features such as, servo motors, forces motors, and so on.

We recommend you read the section, Overview of Feature Creation on page 765 in the chapter, Element Trees: Principles of Feature Creation on page 764. It provides the necessary background for creating features using Creo Parametric TOOLKIT.

Mechanism Spring Feature

A spring generates a translational or rotational spring force in a mechanism. It produces a linear spring force when stretched or compressed, and a torsion force when rotated. The magnitude of the spring force is directly proportional to the amount of displacement from the position of equilibrium.

Springs are created as Creo Parametric features and their values are stored as valid Creo Parametric parameters.

Feature Element Tree for the Mechanism Spring Feature

The element tree for the Mechanism Spring feature is documented in the header file, `ProDamperFeat.h`. The following figure demonstrates the structure of the feature element tree.

Feature Element Tree for Mechanism Spring

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_SPRING_DAMPER_TYPE
   |
   |--PRO_E_SPRING_DAMPER_REF
   |
   |--PRO_E_SPRING_K
   |
   |--PRO_E_SPRING_U
   |
   |--PRO_E_SPRING_DIAMETER
   |
   |--PRO_E_SPRING_USE_DIAMETER
   |
   |--PRO_E_SPRING_ATTACH_POINTS
   |
   |--PRO_E_SPRING_ATTACH_REF
   |
   |--PRO_E_SPRING_FLIP_U_ANGLE
```

The following list details special information about the elements in the feature element tree:

- PRO_E_STD_FEAT_NAME—Specifies the name of the mechanism spring feature.
- PRO_E_SPRING_DAMPER_TYPE—Specifies the mechanism spring feature type. It can have the following values:
  - PRO_SPRING_DAMPER_FORCE—Specifies an extension or compression spring.
  - PRO_SPRING_DAMPER_TORQUE—Specifies a torsion spring.
- PRO_E_SPRING_DAMPER_REF—Specifies the spring placement references. For an extension spring or compression spring, you can select the translational axis or two points on two different bodies as the placement references. For a torsion spring, you can select the rotational axis as the placement reference.
- PRO_E_SPRING_K—Specifies the value for the stiffness coefficient of the spring.
- PRO_E_SPRING_U—Specifies the value for the unstretched length of the spring.
- PRO_E_SPRING_DIAMETER—Specifies the value for the spring icon diameter for an extension spring.
- PRO_E_SPRING_USE_DIAMETER—Specifies the **Adjust Icon Diameter** option (available in the Creo Parametric user interface) that allows you to change the value of the spring icon diameter. This element is not available by default. It takes the following values:
  - PRO_SPRING_USE_DIAMETER_NO
  - PRO_SPRING_USE_DIAMETER_YES
- PRO_E_SPRING_ATTACH_POINTS—Specifies the attachment points for the two ends of a torsion spring. The attachment points can be of the following types:
  - PRO_SPRING_USE_MOTION_AXIS_ZERO—Specifies the JAS (Joint Axis Set) option, where the attachment references are automatically populated with the selected JAS references.
  - PRO_SPRING_CUSTOM_ATTACHMENT_POINTS—Specifies custom attachment references where you can specify two attachment references such as datum planes, datum points, and vertices. The vector between each attachment reference and the selected rotational axis is automatically calculated. By default, the greatest angle between the two vectors is defined as the spring angle. On selecting both the attachment references, the unstretched value of the torsion spring is automatically updated according to the current angle between the two vectors.

- PRO_E_SPRING_ATTACH_REF—Specifies the actual attachment references for the torsion spring depending upon the attachment type set for the element PRO_E_SPRING_ATTACH_POINTS.
- PRO_E_SPRING_FLIP_U_ANGLE—Specifies the option to flip the direction of the current angle between the attachment references for a torsion spring.

Mechanism Damper Feature

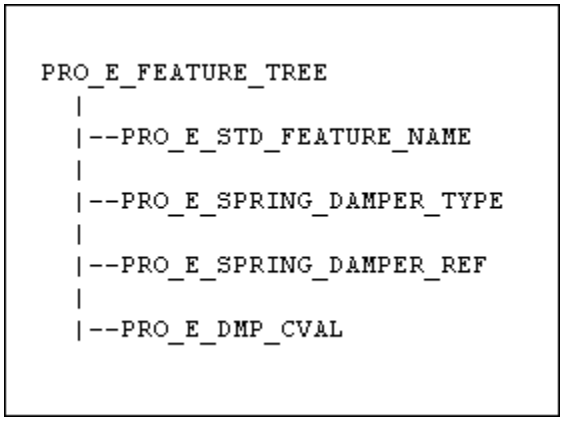
A damper generates a force that removes energy from a moving mechanism and dampens its motion. The damper force is always proportional to the magnitude of velocity of the entity on which you are applying the damper, and acts in the direction opposite to movement.

Dampers are created as Creo Parametric features and their values are stored as valid Creo Parametric parameters.

Feature Element Tree for the Mechanism Damper Feature

The element tree for the Mechanism Damper feature is documented in the header file, ProDamperFeat.h. The following figure demonstrates the structure of the feature element tree.

Feature Element Tree for Mechanism Damper



The following list details special information about the elements in the feature element tree:

- PRO_E_STD_FEAT_NAME—Specifies the name of the mechanism damper feature.
- PRO_E_SPRING_DAMPER_TYPE—Specifies the mechanism damper feature type. It can have the following values:

    - PRO_SPRING_DAMPER_FORCE—Specifies an extension damper or compression damper.
    - PRO_SPRING_DAMPER_TORQUE—Specifies a torsion damper.
- PRO_E_SPRING_DAMPER_REF—Specifies the damper placement references. For an extension damper or compression damper, you can select the translational or slot axis, or two points on two different bodies as the placement references. For a torsion damper, you can select the rotational axis as the placement reference.
- PRO_E_DMP_CVAL—Specifies the value for the damping coefficient.

Mechanism Belt Feature

A belt is treated as a connection connecting all the pulley bodies. The belt provides the same angular velocity to all pulleys connected by it.

Belts are created as Creo Parametric features and their values are stored as valid Creo Parametric parameters.

Feature Element Tree for the Mechanism Belt Feature

The element tree for the Mechanism Belt feature is documented in the header file, ProBeltFeat.h. The following figure demonstrates the structure of the feature element tree.

Feature Element Tree for Mechanism Belt

```
PRO_E_FEATURE_TREE
   |
   |--PRO_E_FEATURE_TYPE
   |--PRO_E_STD_FEATURE_NAME
   |
   |--PRO_E_BELT_PULLEYS
   |     |
   |     |--PRO_E_BELT_PULLEY_RECORD
   |           |--PRO_E_BELT_PULLEY_SEL
   |           |--PRO_E_BELT_PULLEY_WRAP_SIDE
   |           |--PRO_E_BELT_PULLEY_DIAMETER
   |           |--PRO_E_BELT_PULLEY_DIAM_COINCIDENT
   |           |--PRO_E_BELT_PULLEY_CONN_NUM
   |           |--PRO_E_BELT_PULLEY_FLIP_CONN_BODIES
   |           |--PRO_E_BELT_PULLEY_NUM_WRAPS
   |
   |--PRO_E_BELT_PLANE
   |
   |--PRO_E_BELT_DEFINE_CUSTOM_U_LENGTH
   |--PRO_E_BELT_UNSTRETCHED_LENGTH
   |--PRO_E_BELT_STIFFNESS_COEFF
```

The following list details special information about the elements in the feature element tree:

- `PRO_E_FEATURE_TYPE`—Specifies the feature type.
- `PRO_E_STD_FEATURE_NAME`—Specifies the name of the mechanism belt feature.
- `PRO_E_BELT_PULLEYS`—Specifies an array of pulley bodies of the type `PRO_E_BELT_PULLEY_RECORD` which consists of the following elements:
  - `PRO_E_BELT_PULLEY_SEL`—Specifies the geometric reference selected as the pulley body. It can be a cylindrical surface, a circular curve or edge, or a PIN or Cylinder connection. You must define at least two pulley bodies for the belt.

    Creo Parametric automatically selects both the sides of the cylindrical surface or the circular curve or edge so that the pulley is complete. In case of a cylindrical or circular reference, the pulley's axis of rotation is automatically detected by detecting a PIN or Cylinder connection aligned to the theoretical axis of the geometric reference and perpendicular to the belt plane. If such as a connection does not exist, the reference is considered invalid. In case of a connection reference, the rotation axis of the selected connection is used as the pulley axis.

- PRO_E_BELT_PULLEY_WRAP_SIDE—Specifies the pulley wrapping direction. It can be left (-1) or right (+1) and relative to the previous pulley. You can flip the belt to the other contact point detected between the belt and the pulley.
- PRO_E_BELT_PULLEY_DIAMETER—Specifies the pulley diameter. A cylindrical curve with the specified diameter is displayed around the selected pulley reference and on the belt plane. The pulley diameter is coincident with the pulley reference by default, except in case of connection references.
- PRO_E_BELT_PULLEY_DIAM_COINCIDENT—Indicates if the pulley diameter is coincident with the pulley reference. This element is set to PRO_B_TRUE, which means the pulley diameter is set to the value Coincident by default for non-connection references.
- PRO_E_BELT_PULLEY_CONN_NUM—Specifies the number of the PIN or Cylinder connection selected as the pulley reference from the available valid connections. One of the bodies that defines the connection is considered the pulley body, while the other body connected to it is considered the carrier body.
- PRO_E_BELT_PULLEY_FLIP_CONN_BODIES—Specifies the option to flip between the pulley and carrier bodies, in case of connection references. This option is not available in case of geometric references.
- PRO_E_BELT_PULLEY_NUM_WRAPS—Specifies the number of full wraps around the currently selected pulley. The wraps are considered as not overlapping and are not displayed in the 3D icon for the belt. The number of wraps affects the belt length. In case of open ended belts, the number of wraps influences the motion extent of the connected bodies.

- PRO_E_BELT_PLANE—Specifies the planar surface or datum plane that defines the belt plane. The plane should be perpendicular to the rotation axis of the first pulley. This element is optional. If the belt plane is not specified, it is detected based on the selected pulley references as follows:
  - Cylindrical surfaces—If a cylindrical surface is used as the pulley reference, the perpendicular plane positioned in the center of the cylindrical surface (for the first pulley in case of multiple pulleys) is used as the belt plane
  - Circular edge or curve—If a circular edge or curve is used as the pulley reference, the plane for the corresponding edge or curve (for the first pulley in case of multiple pulleys) is used as the belt plane.
  - PIN/Cylinder connections—In case of connection references, the internal orange body zero point (for the first pulley in case of multiple pulleys) is used as the belt plane.

- PRO_E_BELT_DEFINE_CUSTOM_U_LENGTH—Identifies if the unstretched belt length can be specified by the user, or if it is system-defined. The length is system-defined by default.
- PRO_E_BELT_UNSTRETCHED_LENGTH—Specifies the value for the unstretched belt length. The system-defined length is calculated based on the pulley references and their specified diameters. When the user enters a desired belt length, Creo Parametric tries to reconnect the assembly according to the specified belt length. If the reconnect operation fails, the belt length reverts to the previously entered value.
- PRO_E_BELT_STIFFNESS_COEFF—Specifies the belt stiffness coefficient value.

Mechanism 3D Contact Feature

3D contacts are connections between a group of surfaces selected from two parts. These connections are similar to cams, joints, gears, and belts. Depending on the surfaces selected, 3D contacts can be of the following types:

- Sphere to Sphere (point contact)
- Sphere to Planar surface (point contact)
- Cylinder to Cylinder (line contact)
- Cylinder to Planar surface (line contact)
- Sphere to Cylinder (point contact)
- Toroid to Plane (point contact)

3D Contacts are created as Creo Parametric features and their values are stored as Creo Parametric feature parameters.

Feature Element Tree for the Mechanism 3D Contact Feature

The element tree for the Mechanism 3D Contact feature is documented in the header file, ProContact3dFeat.h. The following figure demonstrates the structure of the feature element tree.

Feature Element Tree for Mechanism 3D Contact

```
PRO_E_FEATURE_TREE
  |--PRO_E_FEATURE_TYPE
  |--PRO_E_STD_FEATURE_NAME
  |--PRO_E_C3D_MAT_OPTION1
  |--PRO_E_C3D_REF1_RECS
  |     |
  |     |--PRO_E_C3D_REF_REC
  |           |
  |           |--PRO_E_C3D_REF
  |           |--PRO_E_C3D_REF_FULL_GEOM
  |           |--PRO_E_C3D_REF_FLIP
  |
  |--PRO_E_C3D_MAT_NAME1
  |--PRO_E_C3D_POISSON1
  |--PRO_E_C3D_YOUNG1
  |--PRO_E_C3D_DAMPING1
  |--PRO_E_C3D_MAT_OPTION2
  |--PRO_E_C3D_REF2_RECS
  |     |
  |     |--PRO_E_C3D_REF_REC
  |           |
  |           |--PRO_E_C3D_REF
  |           |--PRO_E_C3D_REF_FULL_GEOM
  |           |--PRO_E_C3D_REF_FLIP
  |
  |--PRO_E_C3D_MAT_NAME2
  |--PRO_E_C3D_POISSON2
  |--PRO_E_C3D_YOUNG2
  |--PRO_E_C3D_DAMPING2
  |--PRO_E_C3D_VERT_RAD
  |--PRO_E_C3D_FRICTION
  |--PRO_E_C3D_STATIC_FRIC_COEF
  |--PRO_E_C3D_KINEM_FRIC_COEF
```

Note

From Pro/ENGINEER Wildfire 5.0 onwards, the feature element tree for the 3D Contact feature has been updated. 3D Contact features created using the old tree are represented in the new tree format. You will need to rebuild your existing Pro/TOOLKIT applications according to the new element tree.

The following list details special information about the elements in the feature element tree:

- PRO_E_FEATURE_TYPE—Specifies the feature type.
- PRO_E_STD_FEATURE_NAME—Specifies the name of the mechanism 3D contact feature.
- PRO_E_C3D_MAT_OPTION1—Specifies the material type for the first contact part. The material type is given by the enumerated type ProC3dMaterialType that takes the following values:
  - PRO_C3D_MAT_DEFAULT—Specifies the material properties of the participating contact part. This is the default type. If this type is set, the elements PRO_E_C3D_POISSON1, PRO_E_C3D_YOUNG1, and PRO_E_C3D_DAMPING1 are not available for modification and are set by default.
  - PRO_C3D_MAT_SEL_MAT—Allows you to select a material type from the list of materials used in the assembly or from the material library directory. If this type is set, the elements PRO_E_C3D_POISSON1, PRO_E_C3D_YOUNG1, and PRO_E_C3D_DAMPING1 are not available for modification and are set as per the selected material.
  - PRO_C3D_MAT_USE_VALS—Allows you to specify the values for the material properties. The elements PRO_E_C3D_POISSON1, PRO_E_C3D_YOUNG1, and PRO_E_C3D_DAMPING1 are available and can be set as required.
- PRO_E_C3D_REF1_RECS—Specifies an array of selected surface references belonging to the first part that is used in the 3D contact. You can select multiple surfaces having a common center, equal diameter and a common edge. The surface reference is of type PRO_E_C3D_REF_REC and consists of the following elements:
  - PRO_E_C3D_REF—Specifies the selected surface reference. The reference can be a spherical, cylindrical, toroidal, or planar surface, or a vertex. If you select a vertex from the first part as one of the references, a sphere is displayed around the vertex and the vertex is considered as a sphere in the 3D contact. If the first reference is a vertex, specify the value for the vertex radius using the element PRO_E_C3D_VERT_RAD. The first surface determines the second part that can be selected for the connection.
  - PRO_E_C3D_REF_FULL_GEOM—Specifies whether a complete surface or a segment of the surface is selected. This option is specified by the enumerated type ProC3dFullGeomFlag and has the following values:
    - PRO_C3D_FULL_GEOM—Specifies that a complete surface has been selected for the 3D contact.

        - PRO_C3D_PARTIAL_GEOM—Specifies that a segment of a cylinder, sphere, or toroid has been selected for the 3D contact.
    - PRO_E_C3D_REF_FLIP—Specifies if the direction for the 3D contact is flipped. The value for this element is PRO_B_FALSE by default which means the contact direction is not flipped and the surface direction is used. This element is applicable only if you use quilt surfaces to create the 3D contacts.
- PRO_E_C3D_MAT_NAME1—Specifies the name of the material type selected for the first contact part. This element can be set only if the element PRO_E_C3D_MAT_OPTION1 is set to PRO_C3D_MAT_SEL_MAT.
- PRO_E_C3D_POISSON1—Specifies the value for Poisson's ratio for the first contact part.
- PRO_E_C3D_YOUNG1—Specifies the value for Young's modulus for the first contact part.
- PRO_E_C3D_DAMPING1—Specifies the value for the damping coefficient for the first contact part.
- PRO_E_C3D_MAT_OPTION2—Specifies the material type for the second contact part. Refer to the description of the element PRO_E_C3D_MAT_OPTION1 for the material types that you can select.
- PRO_E_C3D_REF2_RECS—Specifies an array of selected surface references of the type PRO_E_C3D_REF_REC from the second part that is used in the 3D contact. Refer to the description of the element PRO_E_C3D_REF1_RECS for information on the options that can be set for each surface reference. If you select a vertex from the second part as one of the references, a sphere is displayed around the vertex and the vertex is considered as a sphere in the 3D contact. If the second reference is a vertex, the PRO_E_C3D_VERT_RAD element becomes available for you to specify the value for the vertex radius.
- PRO_E_C3D_MAT_NAME2—Specifies the name of the material type selected for the second contact part. This element can be set only if the element PRO_E_C3D_MAT_OPTION2 is set to PRO_C3D_MAT_SEL_MAT.
- PRO_E_C3D_POISSON2—Specifies the value for Poisson's ratio for the second contact part.
- PRO_E_C3D_YOUNG2—Specifies the value for Young's modulus for the second contact part.
- PRO_E_C3D_DAMPING2—Specifies the value for the damping coefficient for the second contact part.
- PRO_E_C3D_VERT_RAD—Specifies the value for the vertex radius if a vertex is selected as one the references.

- PRO_E_C3D_FRICTION—Identifies if friction will be used in the contact calculation. This element is set to PRO_B_FALSE by default which means no friction.
- PRO_E_C3D_STATIC_FRIC_COEF—Specifies the value for the static friction coefficient.
- PRO_E_C3D_KINEM_FRIC_COEF—Specifies the value for the kinematic friction coefficient.

Mechanism Motor Features

The Mechanism Motor element tree enables you to add servo and force motors.

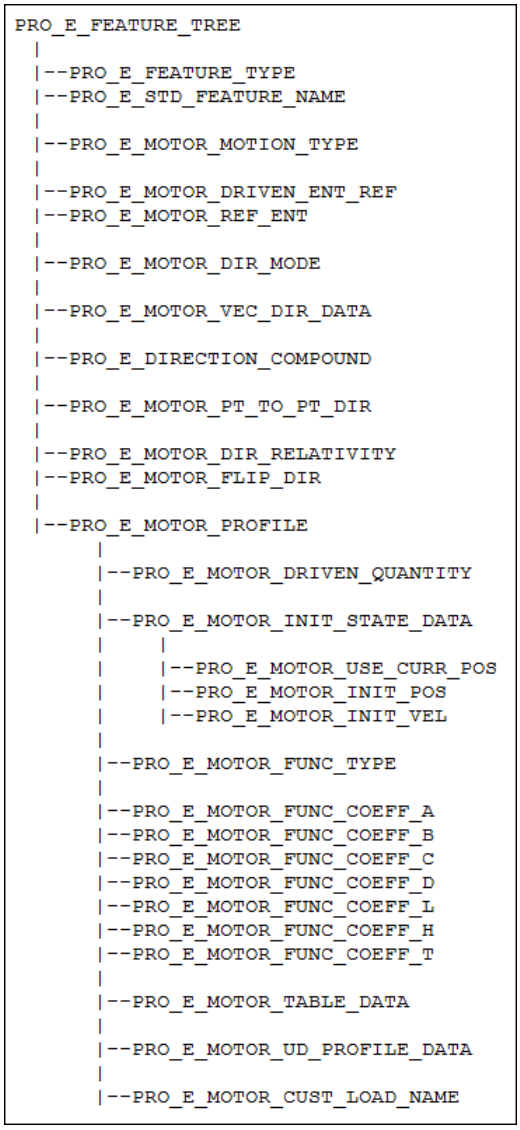
Use servo motors to impose a particular motion on a mechanism. Servo motors cause a specific type of motion to occur between two bodies in a single degree of freedom. Add servo motors to your model to prepare it for analysis.

Use force motors to impose a particular load on a mechanism. A force motor causes motion by applying a force in a single degree of freedom along a translational, rotational, or slot axis.

Feature Element Tree for the Mechanism Motor Feature

The element tree for the Mechanism Motor feature is documented in the header file, ProMotorFeat.h. The following figure demonstrates the structure of the feature element tree.

Feature Element Tree for Mechanism Motor



The following table describes the elements in the element tree for the Mechanism Motor feature:

Element ID	Data Type	Description
PRO_E_FEATURE_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of the motor feature.
PRO_E_STD_FEATURE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the mechanism motor feature.
PRO_E_MOTOR_MOTION_TYPE	PRO_VALUE_TYPE_INT	Specifies the motion type of motor. The motion type is specified using the enumerated data type ProMotorMotionType. The valid values are:

Element ID	Data Type	Description
		• PRO_MOTOR_TRANSLATIONAL • PRO_MOTOR_ROTATIONAL • PRO_MOTOR_SLOT
PRO_E_MOTOR_DRIVEN_ENT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the reference geometry for driven entity. You can select a axes of motion or geometry such as point or plane. **Note** When you select references that are a point or a plane to define the servo motor, you are creating a geometric servo motor.
PRO_E_MOTOR_ENT_REF	PRO_VALUE_TYPE_SELECTION	Specifies the reference geometry for a geometric servo motor. You can select a point or plane.
PRO_E_MOTOR_DIR_MODE	PRO_VALUE_TYPE_INT	Specifies the type of motion direction for motors. The valid values are defined in enumerated data type ProMotorFMDirMode: • PRO_MOTOR_FM_VEC_DIR—The direction is defined by explicit vector in a coordinate system. • PRO_MOTOR_FM_STD_DIR—The direction is defined by standard direction reference such as, straight edge, curve, axis, and plane normal. • PRO_MOTOR_FM_P2P_DIR—The direction is defined by a pair of point or vertex.
PRO_E_MOTOR_VEC_DIR_DATA	PRO_VALUE_TYPE_POINTER	This element is applicable only for geometric force motors. Specifies a compound element which defines the options to set the direction using explicit vector.
PRO_E_MOTOR_VEC_DIR_CSYS	PRO_VALUE_TYPE_SELECTION	Specifies the reference frame for the vector. If the reference frame is not specified, then the World Coordinate System is used.
PRO_E_MOTOR_VEC_DIR_X PRO_E_MOTOR_VEC_DIR_Y PRO_E_MOTOR_VEC_DIR_Z	PRO_VALUE_TYPE_DOUBLE	Specifies the value for X, Y, and Z vectors.

Element ID	Data Type	Description
PRO_E_DIRECTION_COMPOUND	PRO_VALUE_TYPE_POINTER	This element is applicable for geometric servo and force motors. Specifies a compound element which defines the options to set the direction using standard direction reference.
PRO_E_DIRECTION_REFERENCE	PRO_VALUE_TYPE_SELECT	Specifies a motion reference.
PRO_E_DIRECTION_FLIP	PRO_VALUE_TYPE_INT	Flips to reverse the direction of the force or the torque.
PRO_E_MOTOR_PT_TO_PT_DIR	PRO_VALUE_TYPE_SELECT	This element is applicable only for geometric force motors. Specifies the selection of a pair of points to set the direction for point-to-point direction.
PRO_E_MOTOR_DIR_RELATIVITY	PRO_VALUE_TYPE_INT	This element is applicable only for geometric force motors. Specifies the direction of motion relative to ground or driven body.
PRO_E_MOTOR_FLIP_DIR	PRO_VALUE_TYPE_INT	Flips to reverse the direction of the motion. **Note** It defines the direction when connection axis motors or geometric motors do no use the element PRO_E_DIRECTION_COMPOUND to define the direction.
PRO_E_MOTOR_PROFILE	PRO_VALUE_TYPE_POINTER	Specifies a compound element that defines the profile options for a motor.
PRO_E_MOTOR_DRIVEN_QUANTITY	PRO_VALUE_TYPE_INT	Specifies the type of driven quantity. The valid values are defined in the enumerated data type ProMotorDrivenQuantity: • PRO_MOTOR_POSITION—Specifies the motion of servo motor in terms of the position of the selected entity. • PRO_MOTOR_VELOCITY—Specifies the motion of servo motor in terms of its velocity. • PRO_MOTOR_ACCELERATION—Specifies the motion of servo motor in terms of its acceleration. • PRO_MOTOR_FORCE—

Element ID	Data Type	Description
		Specifies a force motor.
PRO_E_MOTOR_INIT_STATE_DATA	PRO_VALUE_TYPE_POINTER	Specifies a compound element that defines the options for initial position of servo motor for PRO_MOTOR_VELOCITY and PRO_MOTOR_ACCELERATION type of motion.
PRO_E_MOTOR_USE_CURR_POS	PRO_VALUE_TYPE_INT	Specifies that the current position of the servo motor is used as the initial starting position.
PRO_E_MOTOR_INIT_POS	PRO_VALUE_TYPE_DOUBLE	This element is applicable only when PRO_E_MOTOR_USE_CURR_POS is set to No. Specifies a starting position for the servo motor.
PRO_E_MOTOR_INIT_VEL	PRO_VALUE_TYPE_DOUBLE	This element is applicable only for PRO_MOTOR_ACCELERATION type of motion. Specifies the initial velocity of the driven entity.
PRO_E_MOTOR_FUNC_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of motion for the motor using the enumerated data type ProMotorFuncType. The valid values are: • PRO_MOTOR_CONSTANT—Creates a constant profile. • PRO_MOTOR_RAMP—Creates a profile that changes linearly over time. • PRO_MOTOR_COSINE—Assigns a cosine wave value to the motor profile. • PRO_MOTOR_SCCA—Simulates a cam profile output. This option is available for acceleration motors only. • PRO_MOTOR_CYCLOIDAL—Simulates a cam profile output. • PRO_MOTOR_PARABOLIC—Simulates a trajectory for a motor. • PRO_MOTOR_POLYNOMIAL—Defines third degree polynomial motor profiles. • PRO_MOTOR_TABLE—Generates the motor motion with values from a four-column table. You can use a

Element ID	Data Type	Description
		table of output measure results. • PRO_MOTOR_USER_DEFINED—Specifies any kind of complex profile defined by multiple expression segments. • PRO_MOTOR_CUSTOM_LOAD—Applies a complex, externally-defined set of loads to your model. This option is only available for the force motor definition.
PRO_E_MOTOR_FUNC_COEFF_A PRO_E_MOTOR_FUNC_COEFF_B PRO_E_MOTOR_FUNC_COEFF_C PRO_E_MOTOR_FUNC_COEFF_D PRO_E_MOTOR_FUNC_COEFF_L PRO_E_MOTOR_FUNC_COEFF_H PRO_E_MOTOR_FUNC_COEFF_T	PRO_VALUE_TYPE_DOUBLE	Specifies the values for the function coefficients.
PRO_E_MOTOR_TABLE_DATA	PRO_VALUE_TYPE_POINTER	Specifies a compound element that defines all the options for table motor type.
PRO_E_MOTOR_TBL_INTERPOL_TYPE	PRO_VALUE_TYPE_INT	Specifies the interpolation method using the enumerated data type ProMotorTableInterpType. The valid values are: • PRO_MOTOR_TBL_LINEAR—Connects the table points with a straight line. • PRO_MOTOR_TBL_SPLINE—Fits a cubic spline to each set of points. • PRO_MOTOR_TBL_MONOTONIC—Produces a monotonic trajectory when you use default velocity values and monotonic magnitude values.
PRO_E_MOTOR_TBL_ROWS	PRO_VALUE_TYPE_POINTER	Specifies an array of table rows.
PRO_E_MOTOR_TBL_ROW	PRO_VALUE_TYPE_POINTER	Specifies a compound element that

Element ID	Data Type	Description
		defines the options for each table row.
PRO_E_MOTOR_TBL_VAR_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the value for independent variables in the first column of the table.
PRO_E_MOTOR_TBL_FUNC_VAL	PRO_VALUE_TYPE_DOUBLE	Specifies the value for driven quantity variables in the second column of the table.
PRO_E_MOTOR_TBL_DERIV_GIVEN	PRO_VALUE_TYPE_INT	This element is applicable only if interpolation type is set to PRO_MOTOR_TBL_MONOTONIC. A flag which checks if derivative value has been specified by the user.
PRO_E_MOTOR_TBL_DERIV_VAL	PRO_VALUE_TYPE_DOUBLE	This element is applicable only if interpolation type is set to PRO_MOTOR_TBL_MONOTONIC. Specifies the value for function derivative in a table row.
PRO_E_MOTOR_TBL_DEPEND_ON_FILE	PRO_VALUE_TYPE_INT	A flag which checks if the table values are dependent on an external file.
PRO_E_MOTOR_TBL_FILE_NAME	PRO_VALUE_TYPE_WSTRING	Specifies the name of the table.
PRO_E_MOTOR_UD_PROFILE_DATA	PRO_VALUE_TYPE_POINTER	Specifies a compound element that defines all the options for user-defined motor type.
PRO_E_MOTOR_UD_EXPR_ARR	PRO_VALUE_TYPE_POINTER	Specifies an array that defines the options for user-defined expressions.
PRO_E_MOTOR_UD_EXPR_DATA	PRO_VALUE_TYPE_POINTER	Specifies a compound that defines the options for each user-defined expression.
PRO_E_MOTOR_UD_EXPR	PRO_VALUE_TYPE_WSTRING	Specifies a user-defined expression.
PRO_E_MOTOR_UD_DOM_TYPE	PRO_VALUE_TYPE_INT	Specifies the type of domain for the expression using the enumerated data type ProMotorUDExprDomainType.
PRO_E_MOTOR_UD_DOM_LOWER_BOUND	PRO_VALUE_TYPE_DOUBLE	Specifies the value for lower bound.

Element ID	Data Type	Description
PRO_E_MOTOR_UD_DOM_UPPER_BOUND	PRO_VALUE_TYPE_DOUBLE	Specifies the value for upper bound.
PRO_E_MOTOR_CUST_LOAD_NAME	PRO_VALUE_TYPE_WSTRING	This element is applicable only for force motors when the element PRO_E_MOTOR_FUNC_TYPE is set to PRO_MOTOR_CUSTOM_LOAD. Specifies the name of the custom file that has pre-defined custom loads.

73

Event-driven Programming: Notifications

Notifications allow your Creo Parametric TOOLKIT application to detect certain types of events in Creo Parametric and provide that your own function is called before or after each such event. Creo Parametric notifies your Creo Parametric TOOLKIT application of these events.

Using Notify

Functions Introduced:

- **ProNotificationSet()**
- **ProNotificationUnset()**

The function `ProNotificationSet()` sets up a notification by specifying the type of event and the pointer to the callback function. The event is specified as a value of the enumerated type `ProNotifyType`. The argument for the callback function has the type `ProFunction`; for consistency, each callback function returns a `ProError` status, even in cases where the status is ignored by Creo Parametric. The callback functions have different arguments, so each callback type has its own typedef that defines its arguments and their types. When calling `ProNotificationSet()`, you must cast the callback function pointer to `ProFunction`.

Note

- If you call `ProNotificationSet()` more than once with the same event type, the existing callback is overwritten with the one supplied in the later call.
- When notifications are set in Creo Parametric TOOLKIT applications, every time an event is triggered, notification messages are added to the trail files. From Creo Parametric 2.0 M210 onward, a new environment variable `PROTK_LOG_DISABLE` enables you to disable this behavior. When set to `true`, the notifications messages are not added to the trail files.

To cancel a notification, call `ProNotificationUnset()`.

Notification Types

The notification events in Creo Parametric fall into the following classes:

File Management Events

Notifications to all of the file management operations in Creo Parametric, such as save, retrieve, copy, rename and so on.

Note

From Creo Parametric 3.0 onward, some callback functions and events for notifications have been deprecated, and will be obsolete in future releases. Refer to the header files `ProMdl.h` and `ProNotify.h` for more information.

The possible file management notifications fall into the following subclasses:

Pre-file Management Events

Your callback function is called before the file management event. It is called only for models that are the explicit objects of the file management operation. For example, it is not called when a part is saved as a result of saving a parent assembly.

If the Creo Parametric user initiated the event, the callback is called before the prompt asking the user for the name of the Creo Parametric models on which to act.

The callback function can optionally write output arguments that determine the Creo Parametric models on which the event will operate. In this case, Creo Parametric will not prompt the user.

The callback function can, by returning an error status, cancel the file management event altogether.

Pre-file Management Events

Event type	Callback typedef	Include file
PRO_FILE_OPEN_OK	ProFileOpenOKAction	ProNotify.h
PRO_MODEL_SAVE_PRE	ProModelSavePreAction	ProMdl.h
PRO_MODEL_COPY_PRE	ProModelCopyPreAction	ProMdl.h
PRO_MODEL_RENAME_PRE	ProModelRenamePreAction	ProMdl.h
PRO_MODEL_RETRIEVE_PRE	ProModelRetrievePreAction	ProMdl.h
PRO_MDL_ERASE_PRE	ProMdlErasePreAction	ProMdl.h
PRO_MDL_PURGE_PRE	ProMdlPurgePreAction	ProMdl.h
PRO_MDL_DELETE_PRE	ProMdlDeletePreAction	ProMdl.h
PRO_MDL_CREATE_PRE	ProMdlCreatePreAction	ProMdl.h
PRO_MDL_START	ProMdlStartAction	ProMdl.h
PRO_CHECKIN_UI_PRE	ProCheckinUIPreAction	ProNotify.h
PRO_MODEL_SAVE_PRE_ALL	ProModelSavePreAllAction	ProMdl.h

The callback function `ProMdlRetrievePreAction` gets called only when you click **File ▸ Open** button in the Creo Parametric user interface. Retrieval of the models by dragging or by double clicking in browser window results in the not triggering the call back . Once set, this notification blocks the Creo Parametric's standard **File ▸ Open** event. Hence, you must substitute the blocked event with your own event through this function.

Pre-All File Management Events

Your callback function is called before all file management events on models, even if those models were not explicitly specified by the user. For example, if you save an assembly, the callback function is also called for any modified parts that will be saved as a result of that action.

Event type	Callback typedef	Include file
PRO_MODEL_SAVE_PRE_ALL	ProModelSavePreAllAction	ProMdl.h

Post-file Management Events

Your callback function is called after the file management operation, and is given input arguments that show which models were the subject of the operation. Like a pre-file management callback, it is called only for models that are the explicit objects of the file management operation.

Post-file Management Events

Event type	Callback typedef	Include file
PRO_MODEL_SAVE_POST	ProModelSavePostAction	ProMdl.h
PRO_MODEL_COPY_POST	ProModelCopyPostAction	ProMdl.h
PRO_MODEL_RENAME_POST	ProModelRenamePostAction	ProMdl.h
PRO_MODEL_ERASE_POST	ProModelErasePostAction	ProMdl.h
PRO_MODEL_RETRIEVE_POST	ProModelRetrievePostAction	ProMdl.h
PRO_MDL_PURGE_POST	ProMdlPurgePostAction	ProMdl.h
PRO_MDL_DELETE_POST	ProMdlDeletePostAction	ProMdl.h
PRO_MDL_CREATE_POST	ProMdlCreatePostAction	ProMdl.h
PRO_UDF_LIB_COMPLETE_POST	ProUdfLibraryCompletePostAction	ProUdf.h

Post All File Management Events

Your the callback function is called for all file management events on models, even if those models were not explicitly specified by the user. For example, if you save an assembly, the callback function is also called for any modified parts that are saved as a result of that action.

Post All File Management Events

Event type	Callback typedef	Include file
PRO_MODEL_SAVE_POST_ALL	ProModelSavePostAllAction	ProMdl.h
PRO_MODEL_COPY_POST_ALL	ProModelCopyPostAllAction	ProMdl.h
PRO_MODEL_ERASE_POST_ALL	ProModelErasePostAllAction	ProMdl.h
PRO_MODEL_RETRIEVE_POST_ALL	ProModelRetrievePostAllAction	ProMdl.h
PRO_MDL_DELETE_POST_ALL	ProMdlDeletePostAllAction	ProMdl.h

File Management Failure Events

Your callback function is called after a file management operation that failed. The function is called with arguments that show the type of file management operation that failed, the models it was operating on, and the type of error encountered (in the form of a ProError value).

Note

From Creo Parametric 3.0 onward, some callback functions and events for notifications have been deprecated. Refer to the header files ProMdl.h and ProNotify.h for more information.

File Management Failed Events

Event type	Callback typedef	Include file
PRO_MODEL_DBMS_FAILURE	ProModelDbmsFailureAction	ProMdl.h
PRO_MDL_CREATE_CANCEL	ProMdlCreateCancelAction	ProNotify.h

Model and Feature Modification Events

Notifications that signal a change to the content of the model. They often include both pre- and post-notifications of change events and include operations on features, solids, parameters and dimensions. See the section Notes on Regeneration Events on page 2015 for more information.

Use the notification types PRO_FEATURE_REROUTE_PRE and PRO_FEATURE_REROUTE_POST to trap the command Feature Reroute in parts or assemblies. Use notification types PRO_FEATURE_REPLACE_PRE and PRO_

FEATURE_REPLACE_POST to trap replacement of assembly components performed in assembly mode with the command Component, Adv Utils, Replace.

The notification types PRO_ASMCOMP_ACTIVATE_PRE and PRO_ASMCOMP_ACTIVATE_POST are called before and after an assembly component is activated within the context of an assembly in Creo Parametric using the right mouse button **Activate**. The notification types provide the path to the currently active component, allowing applications to understand the context of an action in the assembly.

Notes on Regeneration Events

Notifications which trigger before or after feature regeneration should be used carefully, because your callback function is being called while the regeneration of a solid is in progress. This section describes some of the important information you should keep in mind when using notification for model events.

At the start of model regeneration, Creo Parametric discards all data structures that describe geometry, although the geometry items themselves are retained (to preserve the integer identifiers). This means that although you can still traverse the features (using ProSolidFeatVisit()) and the geometry items in a feature (using ProFeatureGeomitemVisit()), geometry items belonging to features not yet regenerated will have no corresponding OHandles. Therefore, functions such as ProSurfaceInit() and ProEdgeInit() will not work. If you analyze the geometry of the features already regenerated, you will see it as unmodified by the features still to be regenerated.

It is dangerous to attempt modifications to the model or file management operations during a regeneration notification function.

Model Modification Events

Event type	Callback typedef	Include file
PRO_DIM_MODIFY_VALUE_PRE	ProDimModifyValuePreAction	ProDimension.h
PRO_FEATURE_CREATE_PRE	ProFeatureCreatePreAction	ProFeature.h
PRO_FEATURE_CREATE_POST	ProFeatureCreatePostAction	ProFeature.h
PRO_FEATURE_COPY_POST	ProFeatureCopyPostAction	ProFeature.h
PRO_FEATURE_DELETE_PRE	ProFeatureDeletePreAction	ProFeature.h
PRO_FEATURE_DELETE_POST	ProFeatureDeletePostAction	ProFeature.h
PRO_FEATURE_SUPPRESS_PRE	ProFeatureSuppressPreAction	ProFeature.h
PRO_FEATURE_SUPPRESS_POST	ProFeatureSuppressPostAction	ProFeature.h
PRO_FEATURE_REDEFINE_	ProFeatureRedefinePreAc	ProFeature.h

Event type	Callback typedef	Include file
PRE	tion	
PRO_FEATURE_REDEFINE_POST	ProFeatureRedefinePostAction	ProFeature.h
PRO_FEATURE_REGEN_PRE	ProFeatureRegenPreAction	ProFeature.h
PRO_FEATURE_REGEN_POST	ProFeatureRegenPostAction	ProFeature.h
PRO_FEATURE_REGEN_FAILURE	ProFeatureRegenFailureAction	ProFeature.h
PRO_FEATURE_NEEDS_REGEN_GET	ProFeatureNeedsRegenGet	ProFeature.h
PRO_FEATURE_REROUTE_PRE	ProFeatureReroutePreAction	ProFeature.h
PRO_FEATURE_REROUTE_POST	ProFeatureReroutePostAction	ProFeature.h
PRO_FEATURE_REPLACE_PRE	ProFeatureReplacePreAction	ProFeature.h
PRO_FEATURE_REPLACE_POST	ProFeatureReplacePostAction	ProFeature.h
PRO_GROUP_UNGROUP_PRE	ProGroupUngroupPreAction	ProGroup.h
PRO_GROUP_UNGROUP_POST	ProGroupUngroupPostAction	ProGroup.h
PRO_PARAM_CREATE_PRE	ProParameterCreatePreAction	ProParameter.h
PRO_PARAM_CREATE_POST	ProParameterCreatePostAction	ProParameter.h
PRO_PARAM_MODIFY_PRE	ProParameterModifyPreAction	ProParameter.h
PRO_PARAM_MODIFY_POST	ProParameterModifyPostAction	ProParameter.h
PRO_PARAM_DELETE_PRE	ProParameterDeletePreAction	ProParameter.h
PRO_PARAM_DELETE_POST	ProParameterDeletePostAction	ProParameter.h
PRO_SOLID_REGEN_PRE	ProSolidRegeneratePreAction	ProSolid.h
PRO_SOLID_REGEN_POST	ProSolidRegeneratePostAction	ProSolid.h
PRO_SOLID_UNIT_CONVERT_PRE	ProSolidUnitConvertPreAction	ProNotify.h
PRO_SOLID_UNIT_CONVERT_POST	ProSolidUnitConvertPostAction	ProNotify.h
PRO_SOLID_PRINC_SYS_UNITS_RENAMED_POST	ProSolidPrincSysUnitsRenamedPostAction	ProNotify.h
PRO_DWGTABLE_ROW_DELETE_PRE	ProDwgtableRowDeletePreAction	ProNotify.h
PRO_DWGTABLE_ROW_DELETE_POST	ProDwgtableRowDeletePostAction	ProNotify.h

Event type	Callback typedef	Include file
PRO_DWGTABLE_DELETE_PRE	ProDwgtableDeletePreAction	ProNotify.h
PRO_DWGTABLE_DELETE_POST	ProDwgtableDeletePostAction	ProNotify.h

Context Change Events

Notifications called after events that change details in the current Creo Parametric context. They allow your application to leverage the details of these changes as needed.

Session Context Events

Event type	Callback typedef	Include file
PRO_DIRECTORY_CHANGE_POST	ProDirectoryChangePostAction	ProNotify.h
PRO_WINDOW_CHANGE_POST	ProWindowChangePostAction	ProNotify.h
PRO_POPUPMENU_CREATE_POST	ProPopupmenuCreatePostAction	ProPopupmenu.h
PRO_POPUPMENU_DESTROY_PRE	ProPopupmenuDestroyPreAction	ProPopupmenu.h
PRO_WINDOW_VACATE_PRE	ProWindowVacatePreAction	ProWindows.h
PRO_WINDOW_OCCUPY_POST	ProWindowOccupyPostAction	ProWindows.h
PRO_WINDOW_OCCUPY_MODEL_POST	ProWindowOccupyModelPostAction	ProWindows.h
PRO_GLOBAL_INTERF_CALC_POST	ProGlobalInterfCalcPostAction	ProNotify.h
PRO_ASMCOMP_ACTIVATE_PRE	ProAsmcompActivatePreAction	ProNotify.h
PRO_ASMCOMP_ACTIVATE_POST	ProAsmcompActivatePostAction	ProNotify.h

Graphics Events

Notifications before and after the repainting of the current Creo Parametric window. This enables you to overlay your own graphics over the window and ensure that they get refreshed when the Creo Parametric window is repainted, for any reason.

Note

These notifications will be called many times; for this reason, your callback routine should be as optimized as possible to avoid performance penalties. For other techniques that may be used to draw graphics, which will repaint with the Creo Parametric window, refer to the section on Display Lists. Refer to the User Interface: Basic Graphics on page 476 chapter for details.

Graphics Events

Event type	Callback typedef	Include file
PRO_MDL_DISPLAY_PRE	ProMdlDisplayPreAction	ProMdl.h
PRO_MDL_DISPLAY_POST	ProMdlDisplayPostAction	ProMdl.h

NC Output Events

Notification of the output from Creo NC of an operation CL data file, or an Creo NC sequence CL data file. This enables you to perform your own post-processing on these files. The callback functions are called with arguments that provide the name of the file created.

NC Output Events

Event type	Callback typedef	Include file
PRO_NCSEQ_CL_POST	ProNcseqClPostAction	ProNotify.h
PRO_OPER_CL_POST	ProMfgoperClPostAction	ProNotify.h

CL Command Events

Notifications that give you the ability to create auxiliary NC sequences with programmatically created CL commands.

CL Command Events

Event type	Callback typedef	Include file
PRO_NCL_COMMAND_EXPAND	ProClCommandExpandAction	ProClCmd.h
PRO_NCL_COMMAND_GET_LOC	ProClCommandGetLocAction	ProClCmd.h

Mold Layout Events

Notifications that are invoked before entering a corresponding Mold Layout dialog.

Mold Layout UI Events

Event type	Callback typedef	Include file
PRO_RMDT_CREATE_IMM_PRE	ProRmdtCreateImmPreAction	ProRmdt.h
PRO_RMDT_BOUND_BOX_PRE	ProRmdtBoundBoxPreAction	ProRmdt.h
PRO_RMDT_CAV_LAYOUT_PRE	ProRmdtCavLayoutPreAction	ProRmdt.h
PRO_RMDT_CREATE_WP_PRE	ProRmdtCreateWpPreAction	ProRmdt.h
PRO_RMDT_MATERIAL_PRE	ProRmdtMaterialPreAction	ProRmdt.h
PRO_RMDT_MBASE_SELECT_PRE	ProRmdtMBaseSelectPreAction	ProRmdt.h

Weld Events

Notifications that give you the ability to customize the results generated by Creo Parametric when gathering info for weld operations.

Weld Events

Event type	Callback typedef	Include file
PRO_DRAWING_WELD_SYMPATH_GET	ProDrawingWeldSympathGetAction	ProNotify.h
PRO_DRAWING_WELD_GROUPIDS_GET	ProDrawingWeldGroupidsGetAction	ProNotify.h
PRO_DRAWING_WELD_SYMTEXT_GET	ProDrawingWeldSymtextGetAction	ProNotify.h

74

Event-driven Programming: External Objects

This chapter describes the Creo Parametric TOOLKIT functions that enable you to create and manipulate external objects.

Summary of External Objects

External objects are objects created by an application that is external to Creo Parametric. Although these objects can be displayed and selected within a Creo Parametric session, they can not be independently created by Creo Parametric. Using Creo Parametric TOOLKIT functions, you can define and manipulate external objects, which are then stored in a model database.

Note

External objects are limited to text and wireframe entities. In addition, external objects can be created for parts and assemblies only. That is, external objects can be stored in a part or assembly database only.

In a Creo Parametric TOOLKIT application, an external object is defined by a `ProExtobj` object. This DHandle identifies an external object in the Creo Parametric database, which contains the following information for the object:

- Object class—A class of external objects is a group that contains objects with similar characteristics. All external objects must belong to a class. Object class is contained in the `ProExtobjClass` object.
- Object data—The object data contains information about the display and selection of an external object. Object data is contained in the `ProWExtobjdata` object.
- Object parameters—External objects can own parameters. You can use the `ProParameter` API to get, set, and modify external object parameters.
- Object references—External objects can reference any Creo Parametric object. This functionality is useful when changes to Creo Parametric objects need to instigate changes in the external objects. The changes are communicated back to your Creo Parametric TOOLKIT application via the callback functions.
- Callback functions—Creo Parametric TOOLKIT enables you to specify callback functions for a class of external objects. These functions are called whenever the external object owner or reference is deleted, suppressed, or modified. In this manner, the appearance and behavior of your external objects can depend on the object owner or reference.

External Objects and Object Classes

This section describes the Creo Parametric TOOLKIT functions that relate to the creation and manipulation of external objects and object classes. Note that this description does not address the display or selection of the external object. For more information on this topic, seeExternal Object Data on page 2024 .

Creating External Object Classes

Functions Introduced:

- **ProExtobjClassCreate()**
- **ProExtobjClassDelete()**

Every external object must belong to a class. The concept of a “class” enables you to group together external objects that exhibit similar characteristics. In addition, classes permit multiple applications to create external objects without conflict.

The `ProExtobjClass` object contains the name and type of an external object class. PTC recommends that you supply a class name unique to your application. The type of the class is an integer that should vary among the different classes.

To register an external object class, pass a completed `ProExtobjClass` object to the function `ProExtobjClassCreate()`. To unregister a class, call the function `ProExtobjClassDelete()`.

Creating External Objects

Functions Introduced:

- **ProExtobjCreate()**
- **ProExtobjDelete()**
- **ProExtobjClassGet()**

After the object class is registered, you can create the external object by calling the function `ProExtobjCreate()`. This function requires as input the object class and owner of the external object. (Currently, the owner of the external object can be a part or an assembly only.) As output, this function gives a pointer to the handle of the newly created external object.

When the external object is created, it is assigned an integer identifier that is persistent from session to session. The external object is saved as part of the model database and will be available when the model is retrieved next.

To delete an external object, call the function `ProExtobjDelete()`. This function requires as input both the object to be deleted and the class to which it belongs. To determine the class of an external object, call the function `ProExtobjClassGet()`.

External Object Owners

Functions Introduced:

- **ProExtobjOwnerobjGet()**
- **ProExtobjOwnerobjSet()**

The owner of an external object is set during the call to `ProExtobjCreate()`. For example, the “owner” would be the part or assembly where the external object resides.

To determine the owner of an external object, call the function `ProExtobjOwnerobjGet()`. To change the owner, call the function `ProExtobjOwnerobjSet()`.

Recycling External Object Identifiers

Functions Introduced:

- **ProExtobjReusableSet()**
- **ProExtobjReusableGet()**
- **ProExtobjReusableClear()**

By default, the identifier of an external object is not “recycled.” When you delete an external object, its identifier is not freed for reuse by external objects that are subsequently created.

You can override this default behavior using the function `ProExtobjReusableSet()`. This function enables external object identifiers to be recycled. To determine whether external object identifiers are set to be recyclable, call the function `ProExtobjReusableGet()`. To reset to the default behavior (no recycling), call the function `ProExtobjReusableClear()`.

External Object Parameters

As with features and models, external objects can also have user-defined parameters. Although you can specify parameters for an external object, there is no method to retrieve these parameters interactively in Creo Parametric. Therefore, external object parameters are a way to store information in the Creo Parametric model that is not accessible to end-users.

You can convert a `ProExtobj` object to a `ProModelitem` object by casting. After this conversion, you can use the function `ProParameterCreate()` to create parameters for the `ProModelitem` object. See the Core: Parameters on page 210 chapter for more information.

External Types and Identifiers for External Objects

Functions Introduced:

- **ProExtobjExttypeSet()**
- **ProExtobjExttypeGet()**

- **ProExtobjExtidSet()**
- **ProExtobjExtidGet()**

`ProExtobj` is a DHandle that contains the type, identifier, and owner of an external object. This information identifies the external object in the Creo Parametric database.

Some applications might require additional type and identifier information to be assigned to external objects. That is, the type and identifier may need to be independent of those assigned within Creo Parametric.

The function `ProExtobjExttypeSet()` sets an external type for an external object. This function calls `ProParameterCreate()` internally and creates a parameter with the name `EXTOBJ_EXTTYPE`. The function `ProExtobjExttypeGet()` obtains the external type for the specified external object.

The function `ProExtobjExtidSet()` sets an external integer identifier for the specified external object. This function calls `ProParameterCreate()` internally and creates a parameter with name the `EXTOBJ_EXTID`. To get the external identifier for a given external object, call the function `ProExtobjExtidGet()`.

Visiting External Objects

Function Introduced:

- **ProExtobjVisit()**

Using the traversal functions for external objects, you can visit each external object in turn, and perform some action or filtration on it. The function `ProExtobjVisit()` specifies action and filter functions of type `ProExtobjVisitAction()` and `ProExtobjFilterAction()`, respectively.

External Object Data

Simply creating an external object does not allow the object to be displayed or selected in Creo Parametric. For this, you must supply external object data that is used, stored, and retrieved by Creo Parametric. The data is removed from the model database when the external object is deleted.

External object data is described by the opaque workspace handle `ProWExtobjdata`. The functions required to initialize and modify this object are specific to the type of data being created. That is, creating display data requires one set of functions, whereas creating selection data requires another.

Once you have created a `ProWExtobjdata` object, the manipulation of the external object data is independent of its contents: the functions required to add or remove data are the same for both display and selection data.

The following sections describe the Creo Parametric TOOLKIT functions that relate to external object data. The sections are as follows:

Display Data for External Objects

Display data gives information to Creo Parametric about how the external object is to appear in the model window. This data must include the color, scale, line type, and transformation of the external object. In addition, display data can include settings that override the user's ability to zoom and spin the external object.

Note that setting display data does not result in the external object being displayed. To see the object, you must repaint the model window using the function `ProWindowRepaint()`.

Allocating Display Data

Function Introduced:

- **ProDispdatAlloc()**

For display data, the workspace handle `ProWExtobjdata` is allocated using the function `ProDispdatAlloc()`. Because the other Creo Parametric TOOLKIT display data functions require `ProWExtobjdata` as input, you must call `ProDispdatAlloc()` before calling the other functions in this section.

The input for `ProDispdatAlloc()` is the address of a `ProWExtobjdata` object that you declare in your application. You must set this `ProWExtobjdata` object to `NULL` before passing its address to `ProDispdatAlloc()`.

Creating the External Object Entity

Functions Introduced:

- **ProDispdatEntsSet()**
- **ProDispdatEntsGet()**
- **ProDispdatEntsWithColorSet()**

External objects are currently limited to text and wireframe entities. You can specify the entities to be displayed by creating an array of `ProCurvedata` objects that contain that necessary information. `ProCurvedata` is a union of specific entity structures, such as line, arrow, arc, circle, spline, and text. Note that when you specify the entities in the `ProCurvedata` array, the coordinate system used is the default model coordinate system.

After you have created the array of `ProCurvedata` objects, you can add entities to the display data by calling the function `ProDispdatEntsSet()`. Note that `ProDispdatEntsSet()` supports only `PRO_ENT_LINE` and `PRO_ENT_ARC` entities. However, you can draw polygons as multiple lines, and circles as arcs of extent 2 `pi`.

To obtain the entities that make up an external object, call the function `ProDispdatEntsGet()`.

The function `ProDispdatEntsWithColorSet()` sets the display data for a list of entities and the color for each entity. The entities that are supported are:

- `PRO_ENT_LINE`
- `PRO_ENT_ARC`

The entities are specified in the local coordinates of the external object. Use the function `ProDispdatTrfSet()` to transform the local coordinates to model coordinates.

Example 1: Creating an External Object on page 2036 shows how to specify an external object that is composed of line segments.

Transformation of the External Object

Functions Introduced:

- **ProDispdatTrfSet()**
- **ProDispdatTrfGet()**
- **ProExtobjScreentrfGet()**

To a perform a coordinate transformation on an external object, you must set the transformation matrix within the associated display data. To do this, call the function `ProDispdatTrfSet()` and pass the transformation matrix as an input argument. To obtain the transformation matrix contained in a particular set of display data, call the function `ProDispdatTrfGet()`.

Example 1: Creating an External Object on page 2036 implements a transformation from default coordinates to a coordinate system that is dependent on the orientation of a selected surface.

Note

Even if you do not want to transform your external object from the default coordinate system, you must specify a transformation matrix. In this case, pass the identity matrix to `ProDispdatTrfSet()`. If you omit this step, your external object will not be displayed.

To obtain the complete transformation of an object from external object coordinates (default coordinates) to screen coordinates, call the function `ProExtobjScreentrfGet()`.

Note

In the assembly mode, `ProExtobjScreentrfGet()` is applicable for external objects owned only by the top assembly model. Use the function `ProDispdatTrfGet()` to retrieve the transformation of external objects in sub-models in the assembly mode. In the part mode, `ProExtobjScreentrfGet()` is applicable for all objects.

External Object Display Properties

Functions Introduced:

- **ProDispdatPropsSet()**
- **ProDispdatPropsGet()**

By default, when users spin or zoom in on a model, external objects are subjected to the same spin and zoom scale as the model. In addition, by default external objects are always displayed, even if the owner or reference objects are suppressed. Setting external object display properties within display data enables you to change these default behaviors.

The `ProExtobjDispprops` object is an enumerated type that contains the possible settings for display properties. To set any of these properties within display data, create a `ProExtobjDispprops` array that contains your settings and pass this array to the function `ProDispdatPropsSet()`. To determine the display settings for specified display data, call the function `ProDispdatPropsGet()`.

The settings contained in `ProExtobjDispprops` are as follows:

- PRO_EXTOBJ_ZOOM_INVARIANT—Sets the external object to be invariant with the zoom scale or magnification of the model. The object appears the same size at all times.
- PRO_EXTOBJ_SPIN_INVARIANT—Set the external object to be invariant with the spin or orientation of the model. The object has the same orientation at all times.
- PRO_EXTOBJ_BLANKED—Blank the display of the external object. This setting is useful if you want the suppress the external object when the reference or owner objects are suppressed.

External Object Color

Functions Introduced:

- **ProDispdatColorGet()**
- **ProDispdatColorSet()**

The enumerated type `ProColortype` specifies the colors available for external objects. To set the object color within display data, call the function `ProDispdatColorSet()`. To determine the color in the specified display data, use `ProDispdatColorGet()`.

Line Styles for External Objects

Functions Introduced:

- **ProDispdatLinestyleSet()**
- **ProDispdatLinestyleGet()**

The enumerated type `ProLinestyle` specifies the line styles available for external objects. To set the object line style within the display data, call the function `ProDispdatLinestyleSet()`. To determine the line style in the specified display data, use `ProDispdatLinestyleGet()`.

External Object Scale

Functions Introduced:

- **ProDispdatScaleSet()**
- **ProDispdatScaleGet()**

To vary the size of your external object without altering the entities themselves, you must specify an object scale factor as part of the display data. To set the scale factor, call the function `ProDispdatScaleSet()`. To determine the scale factor in the specified display data, use `ProDispdatScaleGet()`.

Example 1: Creating an External Object on page 2036 shows how to set the scale of an object to be dependent on the size of the owner object.

Selection Data for External Objects

Functions Introduced:

- **ProSeldatAlloc()**
- **ProSeldatSelboxesSet()**
- **ProSeldatSelboxesGet()**

You can select external objects using the Creo Parametric TOOLKIT selection function `ProSelect()`, with the selection option `ext_obj`. For this selection to be possible, however, you must designate a set of "hot spots," or selection boxes for the object. These selection boxes indicate locations in which mouse selections will cause the external object to be selected. Selection boxes are specified as part of the external object selection data.

The function `ProSeldatAlloc()` allocates selection data in preparation for the specification of the selection boxes.

A selection box is defined by the pair of points contained in a `ProSelbox` object. The coordinates of the points are specified in the external object's coordinate system (the default coordinates). The line between the points forms the diagonal of the selection box; the edges of the box lie parallel to the coordinate axes of the external object. To set the selection boxes within the selection data, call the function `ProSeldatSelboxesSet()` and pass as input a pointer to a list of `ProSelbox` objects. This enables your external object to have more than one associated selection box.

Note

PTC recommends that the size and arrangement of the selection boxes be dependent on the size and shape of the external object. If the external object is compact and uniformly distributed in all coordinate directions, one selection box will probably suffice.

However, if the external object is distributed nonuniformly, or is interfering with other objects, you must designate more specific locations at which selection should occur.

To obtain the list of selection boxes in a given selection data, call the function `ProSeldatSelboxesGet()`.

The ProSelect() function returns an array of ProSelection objects. To obtain a ProExtobj object from a ProSelection object, call the function ProSelectionModelitemGet() and cast the output ProModelitem directly into ProExtobj. (ProExtobj and ProModelitem are DHandles with identical declarations.)

Selecting the Node from the External Application Tree

The tree created by an external application (SPEOS tree) is similar to the Creo Parametric model tree. Each node of this tree represents an external object that has been created by the application. The external objects could be different types of entities, such as, light sources, light sensors, and so on.

Functions Introduced:

- **ProSelectExternalhighlightRegister()**
- **ProSelectExternalselectionRecord()**

The function ProSelectExternalhighlightRegister() registers the call back functions when you select or deselect a node in the user tree or an object in the graphics window. The notification function ProSelectionStartNotify() is called when the function ProSelect() is activated. It notifies the application about entering ProSelect(). The call back function ProSelectionExtHighlightAct() is called when you select or deselect an external object. The TOOLKIT application will highlight the external object or remove the highlight according to the selection. On clicking a tree node, the application creates a ProSelection object and uses the function ProSelectExternalselectionRecord() to pass it to ProSelect(). The input arguments of this function are:

- *selection*—Specifies the selection object created by an external application.
- *action*—Specifes the type of selection. The valid values are:
  - PRO_SELECT_OVERRIDE—For unmodified selection
  - PRO_SELECT_TOGGLE—For CTRL modified selection.

The function ProSelectionEndNotify() notifies the application on exiting the function ProSelect().

Manipulating External Object Data

Functions Introduced:

- **ProExtobjdataAdd()**
- **ProExtobjdataSet()**
- **ProExtobjdataGet()**

- **ProExtobjdataRemove()**
- **ProExtobjdataFree()**

The previous two sections describe how to create and modify external object data. In the case of both display and selection data, the data creation process results in the opaque workspace handle `ProWExtobjdata`. The functions in this section enable you to manipulate how the external object data relates to the object itself.

To add new data to an external object, pass the data handle `ProWExtobjdata` to the function `ProExtobjdataAdd()`. To set the contents of existing object data, call the function `ProExtobjdataSet()`.

The function `ProExtobjdataGet()` obtains the handle for the display or selection data associated with an external object. To specify which type of data you want to retrieve, pass to this function one of the values in the enumerated type `ProExtobjdataType`. The declaration is as follows:

```
typedef enum
{
  PRO_EXTOBJDAT_DISPLAY,
  PRO_EXTOBJDAT_SELBOX
} ProExtobjdataType;
```

To remove data from an external object, use the function `ProExtobjdataRemove()`. To free the memory occupied by external object data, call the function `ProExtobjdataFree()`.

External Object References

You can use external object references to make external objects dependent on model geometry. For example, consider an external object that is modeled as the outward-pointing normal of a surface. Defining the surface as a reference enables the external object to behave appropriately when the surface is modified, deleted, or suppressed.

In general, an external object can reference any of the geometry that belongs to its owner. In addition, if the owner belongs to an assembly, the external object can also reference the geometry of other assembly components, provided that you supply a valid component path.

Note

Setting up the references for an external object does not fully define the dependency between the object and the reference. You must also specify the callback function to be called when some action is taken on the reference.

Creating External Object References

Functions Introduced:

- **ProExtobjRefAlloc()**
- **ProExtobjRefFree()**
- **ProExtobjRefselectionSet()**
- **ProExtobjRefselectionGet()**
- **ProExtobjReftypeSet()**
- **ProExtobjReftypeGet()**
- **ProExtobjRefAdd()**
- **ProExtobjRefRemove()**

The `ProWExtobjRef` object is an opaque workspace handle that defines an external object reference. To allocate the memory for a new external object reference, call the function `ProExtobjRefAlloc()`. To free the memory occupied by an object reference, call the function `ProExtobjRefFree()`.

If you have the `ProSelection` object that corresponds to your intended reference geometry, you can set this `ProSelection` to be the reference by calling the function `ProExtobjRefselectionSet()`. To obtain the `ProSelection` object for a specified reference, use `ProExtobjRefselectionGet()`.

You might need to use “reference types” to differentiate among the references of an external object. To set a reference type, call the function `ProExtobjReftypeSet()`. To obtain the reference type of the specified reference, call the function `ProExtobjReftypeGet()`.

Once you have set the `ProSelection` and the reference type for an external object reference, you must add the reference to the external object using the function `ProExtobjRefAdd()`. To remove a reference from an external object, use `ProExtobjRefRemove()`.

Visiting External Object References

Function Introduced:

- **ProExtobjRefVisit()**

Using the traversal functions for external object references, you can visit each external object reference in turn, and perform some action or filtration on it. The function `ProExtobjRefVisit()` specifies action and filter functions of type `ProExtobjRefVisitAction()` and `ProExtobjRefFilterAction()`, respectively.

Callbacks for External Objects

Functions Introduced:

- **ProExtobjCBAct()**
- **ProExtobjCBEnable()**
- **ProExtobjCallbacksSet()**

External objects are associated with their owners and the references that you specify. Currently, the callbacks mechanism for external objects enables you to receive notification when the reference is deleted, modified, or suppressed. Your callback function can respond in a manner appropriate for the action taken on the reference.

The `ProExtobjCallbacks` object is a structure that specifies the callback functions for each action on the external object's owner or reference. Each callback function is specified by a function pointer of type `ProExtobjCBAct`. When you create an external object class, you should also fill in a `ProExtobjCallbacks` object for that class. To set the callbacks for the class, call the `ProExtobjCallbacksSet()` function.

- Currently, the only supported callbacks for external objects are for deletion, modification, and suppression.
- You cannot use a callback for an external object that references a Creo Parametric feature (and not some geometry of it).

The `ProExtobjCallbacks` data structure is defined as follows:

```
typedef struct
{
  int              enabled_cbs;
  ProExtobjCBAct   display_CB;          /* not yet implemented */
  ProExtobjCBAct   select_CB;           /* not yet implemented */
  ProExtobjCBAct   owner_modify_CB;     /* not yet implemented */
  ProExtobjCBAct   owner_suppress_CB;   /* not yet implemented */
  ProExtobjCBAct   owner_delete_CB;     /* not yet implemented */
  ProExtobjCBAct   ref_modify_CB;
  ProExtobjCBAct   ref_suppress_CB;
  ProExtobjCBAct   ref_delete_CB;
} ProExtobjCallbacks;
```

The first field, `enabled_cbs`, is a flag that enables and disables the callback functions. Set each of the other fields in the structure to the name of the callback function appropriate for each action. To enable or disable the callback functions for a particular action and object class, call the function `ProExtobjCBEnable()`.

As shown in the previous structure, the external objects callbacks are implemented only for cases where the reference is modified, suppressed, or deleted. For this reason, you must exercise caution when enabling callbacks using `ProExtobjCBEnable()`. One of the inputs of the function is an action

bitmask that specifies which callback actions are to be enabled. The action bitmask is composed of members of the enumerated type ProExtobjAction. The values of the enumerated type are as follows:

```
typedef enum
{
   PRO_EO_ALT_DISPLAY    = (1 << 6),
                           /* alternate display --
                              not implemented */
   PRO_EO_ALT_SELECT     = (1 << 7),
                           /* alternate selection --
                              not implemented */
   PRO_EO_ACT_OWN_MODIF  = (1 << 9),
                           /* not implemented */
   PRO_EO_ACT_OWN_SUPPR  = (1 << 10),
                           /* not implemented */
   PRO_EO_ACT_OWN_DELETE = (1 << 11),
                           /* not implemented */
   PRO_EO_ACT_REF_MODIF  = (1 << 13),
   PRO_EO_ACT_REF_SUPPR  = (1 << 14),
   PRO_EO_ACT_REF_DELETE = (1 << 15)
} ProExtobjAction;
```

The action bitmask must not contain any callback actions that are not supported. Given the comments in the ProExtobjCallbacks structure, the only allowed callback actions are PRO_EO_ACT_REF_MODIF, PRO_EO_ACT_REF_SUPPR, and PRO_EO_ACT_REF_DELETE.

The following table describes the actions given in the ProExtobjCallbacks data structure.

Callback Type	When it is Triggered
display_CB	The external object is displayed. Currently, this is not implemented.
select_CB	The external object is selected. Currently, this is not implemented.
owner_modify_CB	The owner of the external object is modified. Currently, this is not implemented.
owner_suppress_CB	The owner of the external object is suppressed. Currently, this is not implemented.
owner_delete_CB	The owner of the external object is deleted. Currently, this is not implemented.
ref_modify_CB	The reference of the external object is modified.
ref_suppress_CB	The reference of the external object is suppressed.
ref_delete_CB	The reference of the external object is deleted.

Warning Mechanism for External Objects

Functions Introduced:

- **ProExtobjClassWarningEnable()**
- **ProExtobjClassWarningDisable()**
- **ProExtobjWarningEnable()**
- **ProExtobjWarningDisable()**

When users perform some action on the references of an external object, you might want to display a warning message to ask users to confirm the action. Creo Parametric TOOLKIT includes functions that implement such warnings, either for all external objects in a class or for individual external objects. For example, if a user attempts to delete a feature whose geometry is referenced by a class of external objects, the Creo Parametric TOOLKIT warning mechanism, if enabled, would open a warning window that states the potential problem. The users would pick Yes to continue with the deletion, or No to abort the deletion.

To enable the warnings for a class of external objects, call the function `ProExtobjClassWarningEnable()`. Note that all external objects that belong to the class will inherit the enabled warning if they are created subsequent to the call to `ProExtobjClassWarningEnable()`.

The input arguments to `ProExtobjClassWarningEnable()` are the class object `ProExtobjClass` and an action bitmask composed of members of the enumerated type `ProExtobjAction` (described in the section Callbacks for External Objects on page 2033). The action bitmask specifies for which actions the warning is to be displayed. Currently, the only supported actions are `PRO_EO_ACT_REF_SUPPR` (suppression of the reference) and `PRO_EO_ACT_REF_DELETE` (deletion of the reference).

To have the warnings displayed for both reference suppression and deletion, the call to `ProExtobjClassWarningEnable()` would appear as follows:

```
ProExtobjClassWarningEnable (&User_arrow_class,
PRO_EO_ACT_REF_SUPPR|PRO_EO_ACT_REF_DELETE);
```

In this call, `User_arrow_class` is declared as a `ProExtobjClass` (an external object class).

To disable the warnings for a class of external objects, call `ProExtobjClassWarningDisable()`. Note that all external objects that belong to the class will inherit the disabled warning if they are created subsequent to the call to the function `ProExtobjClassWarningDisable()`.

To enable warnings for a single external object (not the entire class), call the function `ProExtobjWarningEnable()`. This function is similar to `ProExtobjClassWarningEnable()` except the first argument for `ProExtobjWarningEnable()` is a pointer to a `ProExtobj` object. To disable the warnings for a single external object, call the function `ProExtobjWarningDisable()`.

Example 1: Creating an External Object

The sample code in the file `UgExtobjCreate.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_extobj` shows how to create an external object at a location specified by the user. The external object is a green arrow that is aligned with the normal to a selected surface.

Note

For the sake of simplicity, the example does not implement selection data.

75

Event-driven Programming: Toolkit-Based Analysis

This chapter describes the functions that enable you to create analysis and analysis feature objects in a Creo Parametric solid.

Overview

Creo Behavioral Modeling allows the creation of two types of objects in a Creo Parametric solid:

- Analysis
- Analysis Feature

In Creo Parametric you can create an analysis using the commands under the **Analysis** tab. Analyses show the results of certain standard types of measurement or calculation, for example, curvature of an edge or surface, or the center of gravity of a solid. Users can name an analysis and store it in the solid, along with the references to the geometry items it analyses. The analysis is then reevaluated automatically upon each model regeneration, and can be queried at any time using the **Analysis** command. Such an analysis is stored separately from the features and geometry items in the solid.

An analysis feature is a feature that uses an analysis to determine the values of its feature parameters and the shape of its geometry items. An analysis feature is a variety of datum feature and is created using the Creo Parametric **Datum** command. An example of the use of an analysis feature is the creation of a coordinate system datum at the center of gravity of a solid, aligned with its axes of inertia. Another example is a pair of datum points at the closest points of two parts in an assembly.

Creo Parametric TOOLKIT analysis functions allow definition of analyses and analysis features whose computations are performed by callback functions provided by the Creo Parametric TOOLKIT application. This means that Creo Parametric TOOLKIT can be used to make analysis computations, and determine feature geometry, in ways not native to Creo Parametric. We refer to the analyses and analysis features defined by Creo Parametric TOOLKIT as toolkit-based; this stresses the fact that the computations could be performed by a separate Creo Parametric TOOLKIT application.

Creo Parametric TOOLKIT users should practice using standard Creo Parametric analyses and analysis features before studying toolkit-based analyses.

The functions and data structures specific to toolkit-based analysis features are declared in the header file `ProAnalysis.h`.

Interactive Creation of Toolkit-Based Analysis

If a Creo Parametric TOOLKIT application registers a Toolkit-Based Analysis type, the **Analysis** menu on the Creo Parametric toolbar contains an extra button labeled **Toolkit-Based**. Click this button to see the **Toolkit-Based** dialog box.

The selector at the top of the dialog box shows the types of external analysis registered by the Creo Parametric TOOLKIT application.

When you have chosen the type, click the **Analysis** UI button; this calls the Creo Parametric TOOLKIT callback, which prompts the user for the information needed by the analysis. When you have answered all the prompts, click **Compute**.

Compute performs the analysis and displays the resulting text, if any, in the text area of the dialog box. It may also display some graphics.

When you have clicked **Compute**, you may also click **Info**, which displays the output text in an information window.

The **Saved Analyses** and **Close** buttons behave as for standard analyses.

Interactive Creation of Toolkit-Based Analysis Feature

In Creo Parametric, under the **Analysis** tab, in the **Manage** group, select the **Analysis** command. This displays the Creo Parametric **ANALYSIS** dialog box, which leads the user through the definition of the elements of the analysis feature. If a Creo Parametric TOOLKIT application is running and has registered at least one type of toolkit-based analysis, there will be an additional button labeled **Toolkit-Based** in the Type section.

Set the name of the analysis first, set the type to Toolkit-Based Analysis, and continue to the next step, which is called “Definition”. The **Toolkit-Based** dialog box appears. This behaves exactly as for an external analysis, until you click the **Close** button. It then returns to the **ANALYSIS** dialog box, to allow you to continue specifying the feature elements.

The remaining elements are Result Params and Result Datums. These behave exactly as for standard analysis features, except that the parameters and datums are defined by the Creo Parametric TOOLKIT application. One or other may be absent if the toolkit-based analysis defines no parameters or no datums.

Storage of Toolkit-Based Analysis Feature in Creo Parametric

A toolkit-based analysis feature is stored in Creo Parametric in exactly the same way as any other feature. It appears in the model tree as a feature of type **Analysis**, and all the regular **Feature** commands can be used on it.

The references to existing geometry that the feature needs to calculate its own parameters and geometry are given to Creo Parametric by the Creo Parametric TOOLKIT application in the form of `ProSelection` structures. Creo Parametric stores these references, using the standard method for storing feature

references. This means that the Creo Parametric TOOLKIT application does not need to store this information, and that the feature automatically has the correct behavior for the following Creo Parametric functions:

- Feature Regeneration—Creo Parametric knows from the feature references what other features it depends on, and therefore whether the features needs to be included in a particular regeneration.
- Rerouting Features
- Patterning Features

The Creo Parametric TOOLKIT application may also give Creo Parametric the values of any variables that control the geometry, and Creo Parametric stores them as feature dimensions in the new feature. The Creo Parametric TOOLKIT application can then read the current dimension values when recomputing the feature geometry. This means the feature correctly responds to dimension changes as a result of, for example, being driven by a relation.

The Creo Parametric TOOLKIT application must store as external data any information about toolkit-based analysis features that is not stored as either geometry references or dimensions.

Registering a Toolkit-Based Analysis with Creo Parametric

Function Introduced:

- **ProAnalysisTypeRegister()**

The function `ProAnalysisTypeRegister()` registers a toolkit-based analysis with Creo Parametric by specifying its type name and the set of callback functions to be used when creating it and performing the computation. Call this function in `user_initialize()`.

If called correctly, both the **Analysis** command and the **ANALYSIS** dialog box used in creating an analysis feature will include the button **Toolkit-Based** as the analysis type. Refer to the sections Interactive Creation of Toolkit-Based Analysis on page 2038 and Interactive Creation of Toolkit-Based Analysis Feature on page 2039 for more information on creating these objects.

Analysis Callbacks

When registering a Toolkit-Based Analysis type, callbacks must be provided for each of the following 13 types:

- `ui`
- `dims`

- `infoalloc`
- `infofree`
- `compcheck`
- `compute`
- `display`
- `output`
- `savecheck`
- `infosave`
- `inforetrieve`
- `infocopy`
- `result`

Each callback is passed an argument of the `ProAnalysis` type, which is an opaque handle and identifies the analysis information stored by Creo Parametric.

The following table explains when the callbacks are called, and how each one should be used.

When Creo Parametric creates a toolkit-based analysis or analysis feature:

Callback	Description
`infoalloc`	Allocate memory for the Creo Parametric TOOLKIT application information about the toolkit-based analysis.
`ui`	Creo Parametric TOOLKIT prompts the user for inputs that define the analysisfor example, select a surface datum point on which to position a csys.
`compcheck`	Tell Creo Parametric whether the computation can be performed. If the Creo Parametric TOOLKIT application cannot perform the computation (for example, because input data is unavailable), it returns an error and the regeneration fails.
`compute`	Perform the analysis computation and store the results in memory.
`display`	Display graphics showing the computation result.
`output`	Pass a set of text lines to Creo Parametric for display in the **ANALYSIS** dialog box to show the result of the computation.
`infocopy`	Copy the application information from an existing analysis to a new one. Call `infocopy` during creation because of the way in which Creo Parametric handles feature creation.

Callback	Description
`dims`	Creo Parametric TOOLKIT gives Creo Parametric a list of double values needed to calculate the geometry. Creo Parametric stores these as model dimensions.
`result`	Creo Parametric TOOLKIT gives Creo Parametric a description of the feature parameters and geometry items that result from the computation of the analysis. Creo Parametric may also call this callback when it needs to know only the number and names of parameters and datums; an example is when the user selects **Feature Info**. For more details, refer to the section Results Data on page 2043.

When the Creo Parametric user saves the analysis to the solid:

Callback	Description
`savecheck`	Tell Creo Parametric whether the description of the analysis can be saved.
`infosave`	Give Creo Parametric a list of geometry items referenced by the analysis. Creo Parametric stores these using its own internal mechanism for storing references. The references appear in the model as feature references, and are used to determine the relationship of the feature to other features, and therefore when the feature needs to be regenerated. OR Store any other data as external data.

When the Creo Parametric user retrieves a solid containing analyses:

Callback	Description
`inforetrieve`	Creo Parametric provides an array of `ProSelection` objects representing the geometry references it stored with the analysis. (This means that the Creo Parametric TOOLKIT application does not need to save these references between sessionsCreo Parametric uses its own mechanism.)

When the Creo Parametric user leaves the **ANALYSIS** dialog box without saving the new analysis, or erases a solid containing a toolkit-based analysis:

Callback	Description
infofree	The Creo Parametric TOOLKIT application frees the memory used by its internal description of the analysis.

Many of the callbacks will be called during other commands in Creo Parametric whenever the toolkit-based analysis or analysis feature is affected.

Creo Parametric TOOLKIT Analysis Information

Functions Introduced:

- **ProAnalysisInfoGet()**
- **ProAnalysisInfoSet()**
- **ProAnalysisTypeGet()**

The Creo Parametric TOOLKIT application must keep its own description of the analysis in memory in order to perform the computation. This memory is allocated, filled, and freed in the callbacks provided, as described in the section Analysis Callbacks on page 2040.

The callback-calling sequence may vary depending upon exactly what the Creo Parametric user does. This means Creo Parametric TOOLKIT applications should not use global variables to pass the “current” analysis from one callback to another.

Instead of such variables, Creo Parametric can store a pointer to the Creo Parametric TOOLKIT data stored inside the `ProAnalysis` object. Use function `ProAnalysisTypeGet()` to return the type of a specified analysis. The `infoalloc` callback should end with a call to `ProAnalysisInfoSet()`, which gives Creo Parametric the pointer to the new description. Every other callback that needs to use this data should call `ProAnalysisInfoGet()` to get a pointer to the application data for the analysis object, rather than assume this by context.

Creo Parametric cannot access the user data, nor can it provide general storage of this data in the Creo Parametric file when the solid is saved. As explained in the section Analysis Callbacks on page 2040 earlier, the callbacks infosave and inforetrieve respectively give to Creo Parametric, and return after retrieval, geometry references contained in the information; double values can be stored as dimensions. Note that any other information the application needs to store should be saved and retrieved inside infosave and inforetrieve using Creo Parametric TOOLKIT external data.

Results Data

This section describes in more detail the data given to Creo Parametric by the Creo Parametric TOOLKIT application as the output from the results callback.

The output consists of two arrays, one for the feature parameters, the other for the feature geometry. Each of these is a `ProArray` allocated by Creo Parametric before calling the callback.

The structure for a feature parameter is:

```
typedef struct analysis_param
```

```
{
    ProName       name;
    ProBoolean    create;
    ProLine       description;
    ProParamvalue *values;
} ProAnalysisParameter;
```

The name is that of the feature parameter that will be created. The `create` flag shows the default setting of the **Create** option for the parameter in the **ANALYSIS** dialog box. The description appears alongside the parameter in the **ANALYSIS** dialog box. The array of values should have only a single item in it in the current release. The structure `ProParamvalue` is the same one used for accessing user parameters through the functions in `ProParameter.h` and `ProParamvalue.h`. The value type must be “double” in the current release.

The structure for a geometry item is:

```
typedef struct analysis_geom
{
    ProName name;
    ProBoolean create;
    ProAnalysisEntityType type;
    ProAnalysisEntity *shapes;
} ProAnalysisGeomitem;
```

The name is given to the resulting datum (NOT to the feature), but with a numerical suffix to ensure that the name is unique in the Creo Parametric model. The `create` flag is exactly as for parameters.

The entity type is an enum which is a subset of the object types in `ProType`. The types supported by this release are:

`PRO_ANALYSIS_CURVE`	Curve
`PRO_ANALYSIS_CSYS`	Coord csys
`PRO_ANALYSIS_POINT`	Datum point
`PRO_ANALYSIS_COMP_CRV`	Composite curve
`PRO_ANALYSIS_SURFACE`	Surface

Array “shapes” contains any number of geometric entities of the same type. The union that represents an entity shape is:

```
typedef union
{
    ProAnalysisSrfData    *surface;
    ProQuiltdata          *quilt;
    ProCurvedata          *curve;
    ProCsysdata           csys;
} ProAnalysisEntity;
```

The fields in this union all have types that are generic geometry types in Creo Parametric TOOLKIT, and are declared in the appropriate headers: `ProSurfacedata.h`, `ProQuiltdata.h`, `ProCurvedata.h`, `ProCsysdata.h`, `ProEdgedata.h`. The first three fields, although they are

pointers, are not opaque: in spawn mode or asynchronous mode they point to memory in the Creo Parametric TOOLKIT process, not in the Creo Parametric process. It is recommended that you build these structures using the functions in the corresponding header files, for example, `ProCurvedataAlloc()`, `ProLinedataInit()`, and so on.

ProAnalysisSrfData Structure

The `ProAnalysisSrfData` structure is used to define a datum surface from a toolkit-based analysis feature. This structure consists of:

```
typedef struct  ProAnalysisSrfData
{
      ProEdgedata    *edge_data;
      ProSurfacedata *pro_surface;
} ProAnalysisSrfData;
```

The `ProSurfacedata*` structure contains the surface shape, parameters, and a populated `ProContourdata` structure referencing the boundary edges. The `ProEdgedata*` member should be a `ProArray` of edge geometric data whose ids are referenced by the contour data in the surface data structure.

Function Introduced:

- **ProAnalysissrfdataAlloc()**

The function `ProAnalysissrfdataAlloc()` allocates the `ProAnalysisSrfData` data structure.

Example 1: Offset Coordinate System Datum

The sample code in the file `UgExtAnalysisSurfcsys.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_ext_analysis` shows a Creo Parametric TOOLKIT application that defines a toolkit-based analysis that builds a coordinate system datum at an offset from a surface point. The Z axis can be an inward or an outward normal to the surface, the X and Y axes are aligned with the UV mesh lines in the surface, and the origin can be offset in the Z direction. The offset value is stored as a feature dimension.

Analysis Attributes

Functions Introduced:

- **ProAnalysisAttrSet()**
- **ProAnalysisAttrIsSet()**

These functions allow you to get and set certain attributes on a toolkit-based analysis. The only attribute defined in the current release is `PROANALYSIS_COMPUTE_OFF`. If this is set, the compute and result callbacks will not be called

during regeneration of the model. If the toolkit-based analysis belongs to a feature, the geometry of the feature will be frozen until the PROANALYSIS_COMPUTE_OFF is unset again. If the feature geometry includes a surface curve, the 3D location of the curve will be recalculated during regeneration in accordance with the existing UV curves definition but using the new geometry of the surface. This means the curve remains in the surface even if the surface moves while COMPUTE_OFF is TRUE.

Use the PROANALYSIS_COMPUTE_OFF attribute to temporarily turn off the toolkit-based analysis to save time when making other changes to the model.

Visiting Saved Toolkit-Based Analyses

Functions Introduced:

- **ProSolidAnalysisVisit()**
- **ProAnalysisNameGet()**

The function ProSolidAnalysisVisit() visits a saved toolkit-based analysis in a part or assembly. It does not visit standard saved analyses, nor toolkit-based analyses in features.

The function ProAnalysisNameGet() provides the name under which a toolkit-based analysis is saved.

Visiting Toolkit-Based Analyses Features

Functions Introduced:

- **ProFeatureAnalysisGet()**

To visit analyses in analysis features, use ProSolidFeatureVisit(), filter for features whose type is PRO_FEAT_ANALYSIS, and call ProFeatureAnalysisGet() on each feature.

Using the Model without Creo Parametric TOOLKIT

If a model contains external analyses or external analysis features of a type defined by a Creo Parametric TOOLKIT application, and that model is retrieved into Creo Parametric while the Creo Parametric TOOLKIT application is not running, Creo Parametric will not attempt to recompute those analyses during regeneration. Saved toolkit-based analyses will retain their old values, and toolkit-based analysis features will have their geometry frozen.

If a Creo Parametric TOOLKIT application is terminated during a Creo Parametric session, any toolkit-based analysis types it registered will be automatically deregistered, and any analyses that use those types will be frozen.

76

Event-driven Programming: Foreign Datum Curves

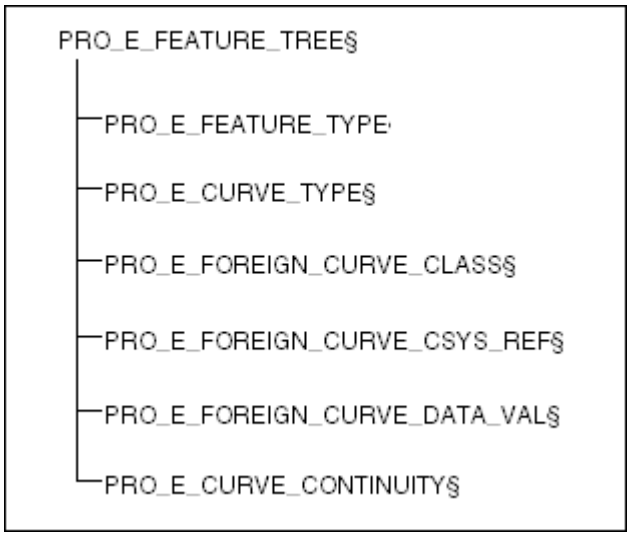
This chapter describes the Creo Parametric TOOLKIT functions that enable you to create foreign datum curves.

Foreign Datum Curves

In Creo Parametric TOOLKIT, you create foreign datum curves using the feature creation techniques described in the chapter Element Trees: Principles of Feature Creation on page 764. The header file `ProForeignCurve.h` contains the element tree structure and a table that maps each element to an element identifier, value type, and valid values.

The following figure shows the element tree structure for foreign datum curve creation. Note that all elements are required.

Element Tree for Foreign Datum Curve



As the element tree implies, foreign datum curve creation requires that you provide the feature type, curve type, curve class, reference coordinate system, data used in the analytical representation of the curve, and curve continuity. Creo Parametric uses this information, together with an evaluation function, to create an internal representation of the curve.

Providing an Evaluation Function

Function Introduced:

- **ProForeignCurveEvalFunction()**

In addition to building the element tree for your datum curve feature, you must provide its analytical representation to Creo Parametric. This representation is made available to Creo Parametric in a special function called an evaluator, or evaluation function.

The evaluation function must contain parameterized equations for the X, Y, and Z coordinates of points that define the curve. If C(X,Y,Z) is a function representing the curve in three-dimensional space, you can represent the parameterized equations for each coordinate as follows:

```
X = f(t)
Y = g(t)
Z = h(t)
```

In these equations, the parameter `t` ranges from 0 to 1 over the extent of the curve. For example, a parametric representation of a circle of radius `R` lying in the XY-plane, whose center coincides with the origin, is as follows:

```
X = R*cos(2*PI*t);
Y = R*sin(2*PI*t);
Z = 0;
```

In these equations, `PI` = 3.14159.

Creo Parametric TOOLKIT provides the prototype for the evaluation function. The syntax is as follows:

```
typedef ProError (*ProForeignCurveEvalFunction)
(
  ProName        class,          /* input */
  wchar_t       *data_string,    /* input */
  ProSelection   csys,           /* input */
  double         curve_param,    /* input */
  ProVector      xyz_point,      /* output */
  ProVector      deriv1,         /* output */
  ProVector      deriv2          /* output */
);
```

The function arguments are as follows:

- *class*—Identifies the type of curves generated by the evaluation function.
- *data_string*—The flag that controls specific attributes of the curve.
- *csys*—The reference coordinate system with respect to which the curve geometry is defined. Pass it to the evaluation function as a `ProSelection` object.
- *curve_param*—The parameter value at which the X, Y, and Z coordinates, as well as the first and second derivatives, will be evaluated.
- *xyz_point*—The X, Y, and Z coordinates at the value of *curve_param*.
- *deriv1*—The values of the first derivatives of X, Y, and Z with respect to the parameter, at the value of *curve_param*.
- *deriv2*—The values of the second derivatives of X, Y, and Z with respect to the parameter, at the value of *curve_param*.

All arguments are passed to the evaluation function by Creo Parametric, based on the values you provide for the elements in the element tree.

A single evaluation function can be used to create a number of curve variations within a given class. The parameterized curve equations typically contain constants whose values control the shape, size, location, and orientation of the curve. You can write the evaluation function such that, depending on the value of the *data_string* argument, different values of those constants will be used to calculate the location of points on the curve.

Curve Continuity

Curve continuity, in a sense, defines the smoothness of intersections between the ends of the foreign curve and other geometry in the model. It also defines the continuity of three-dimensional geometry created from the curve, such as a swept surface. First-order continuity implies that the first derivatives of two adjoining curve segments are equal at the point at which the curves join. Second-order continuity is similarly defined. Depending on the curve continuity you want, the evaluator function needs to contain first and second derivatives of the parameterized curve equations.

You specify the curve continuity using the `PRO_E_CURVE_CONTINUITY` element in the element tree. The valid values, contained in the enumerated type `ProForeignCrvCont`, are as follows:

- `PRO_FOREIGN_CURVE_CALC_XYZ`
- `PRO_FOREIGN_CURVE_CALC_XYZ_1_DER`
- `PRO_FOREIGN_CURVE_CALC_XYZ_1_AND_2_DER`

These values correspond to zeroth-, first-, and second-order continuity, respectively. If you use the value `PRO_FOREIGN_CURVE_CALC_XYZ`, Creo Parametric passes `NULL` for *deriv1* and *deriv2* to the evaluation function. Similarly, if you use the value `PRO_FOREIGN_CURVE_CALC_XYZ_1_DER`, Creo Parametric passes `NULL` for *deriv2* to the evaluation function. Therefore, you should check for `NULL` values of *deriv1* and *deriv2* in your evaluation function before trying to assign derivative values to them.

Creo Parametric calls your evaluation function multiple times for a series of values of the curve parameter, ranging from 0 to 1. The function outputs the following information:

- X, Y, and Z coordinates of the curve at the specified parameter value
- Values of the first and second derivatives, as needed for the desired curve continuity

These values are then used by Creo Parametric to construct the curve.

Binding the Evaluation Function to a Class

Function Introduced:

- **ProForeignCurveClassEvalSet()**

The evaluation function must be bound to a class. This is done with a call to the function `ProForeignCurveClassEvalSet()`. The function takes as arguments the class name and a pointer to the evaluation function. If you call `ProForeignCurveClassEvalSet()` and pass `NULL` for the evaluation function pointer, it unbinds a previously bound evaluation function from the class.

77

Task Based Application Libraries

Applications created using the different Creo Parametric API products are interoperable. These products use Creo Parametric as the medium of interactions, eliminating the task of writing native -platform specific interactions between different programming languages.

ProArgument and Argument Management

Use the data structure `ProArgument` to pass application data to and from tasks in other applications. The declaration for this structure is:

```
typedef struct pro_argument
  {
    ProName      label;
    ProValueData value;
  } ProArgument;
```

The `ProValueData` structure supports the following argument types:

- Integer
- Double
- String (char*)
- String (wchar_t*)
- Boolean
- `ProSelection`
- `ProMatrix`

Do not use the value type `PRO_VALUE_TYPE_POINTER` (provided with this structure in order to support feature element tree values) when passing arguments between applications.

Functions Introduced:

- **ProArgumentByLabelGet()**
- **ProValuedataStringSet()**
- **ProValuedataWstringSet()**
- **ProValuedataTransformGet()**
- **ProValuedataTransformSet()**
- **ProArgumentProarrayFree()**

Use the function `ProArgumentByLabelGet()` to locate an argument within a `ProArray` of `ProArgument` structures passed between applications.

Use the function `ProValuedataStringSet()` to allocate and copy memory into the `ProValuedata` structure for a `char*` argument. Using this function ensures that `ProArgumentProarrayFree()` releases all memory in an arguments array.

Use the function `ProValuedataWstringSet()` to allocate and copy memory into the `ProValuedata` structure for a `wchar_t*` argument. Use this function to ensure that `ProArgumentProarrayFree()` releases all memory in an arguments array.

Use the function ProValuedataTransformSet() to allocate and copy memory into the ProValuedata structure for a ProMatrix argument. Use this function to ensure that ProArgumentProarrayFree() releases all memory in an arguments array.

Use the function ProValuedataTransformGet() to copy a ProMatrix value into a local variable. The matrix data is not directly accessible from the double** member of the ProValuedata structure.

Use the function ProArgumentProarrayFree() to completely free an array of ProArgument structures.

Creating Creo Parametric TOOLKIT DLL Task Libraries

Functions that are intended to act as Creo Parametric TOOLKIT task library functions must have the same signature as ProTkdllFunction():

```
typedef ProError (*ProTkdllFunction) (
ProArgument* inputs, ProArgument** outputs );
```

Use the preprocessor macro PRO_TK_DLL_EXPORT with any function that must be accessible to external applications. This macro provides platform-specific instructions to the compiler to make the function visible to other external applications. This macro must be placed in the function prototype, if it exists, and in the function definition if the prototype does not exist.

Some platforms require externally visible symbols to be declared on the application link line.

Memory Management in Task Library Functions

To avoid memory leaks or overwrites, the DLL functions must use proper memory management. The DLL function must:

- Ensure that the contents of the input ProArgument array are not freed. This is taken care of by the calling application and the Creo Parametric TOOLKIT communications code.
- Use the ProValuedata*Set() functions to assign values to the output ProArgument array. This allows the calling application to free the output array with ProArgumentProarrayFree().

Example 1: An Exported Toolkit Task Function

The sample code in the file UgImportfeatCreate.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_featcreat, function demonstrates how you can set up and implement a Creo

Parametric TOOLKIT task function accessible from other Creo Parametric auxiliary applications. It uses the required signature and macro for task functions. It parses the input argument array to search for parameters required by the task.

Launching Creo Parametric TOOLKIT DLL Functions

Functions Introduced:

- **ProToolkitDllLoad()**
- **ProToolkitTaskExecute()**
- **ProToolkitDllUnload()**
- **ProToolkitDllIdGet()**
- **ProToolkitDllHandleGet()**

Use the function `ProToolkitDllLoad()` to register and start a Creo Parametric TOOLKIT DLL. The input parameters of this function are similar to the fields of a registry file and are as follows:

- *app_name*—The name of the application to initialize.
- *exec_file*—The DLL file to load, including its full path.
- *text_dir*—The path to the application's message and UI text files.
- *user_display*—Set this parameter to `PRO_B_TRUE` if you want the interactive user to be able to see the application registered in the Creo Parametric User Interface and to see error messages if the application fails.

The function outputs a handle to the loaded DLL in the form of a `ProToolkitDllHandle`. If the function fails to load the DLL, the function outputs information describing the failure and the application's `user_initialize()` function is called.

Use the function `ProToolkitTaskExecute()` to call a properly designated function of the Creo Parametric TOOLKIT DLL library. You can pass arbitrary combinations of input arguments to the library function.

Use the function `ProToolkitDllUnload()` to shutdown a Creo Parametric TOOLKIT DLL previously loaded by `ProToolkitDllLoad()`. The application's `user_terminate()` function is called.

Use the function `ProToolkitDllIdGet()` to get a string representation of the DLL application. The string representation can be sent to other applications, which can use `ProToolkitDllHandleGet()` to obtain the Creo Parametric TOOLKIT DLL handle using this string representation. Pass `NULL` to the first argument of `ProToolkitDllIdGet()` to get the identifier for the calling application.

Custom Creo Parametric TOOLKIT DLL Tasks for Creo Distributed Batch

You can create a customized Creo Distributed Batch task by providing a Creo Parametric TOOLKIT DLL to be run by the Creo Distributed Batch service. The steps required to create and deploy a custom DLL task for Creo Distributed Batch are as follows:

- Compose the Creo Parametric TOOLKIT DLL containing a function that meets the criteria for a custom DLL task function. Refer to the Creating Creo Parametric TOOLKIT DLL Task Libraries on page 2055 section for the function signature. Use the standard Creo Distributed Batch arguments while coding the function. Refer to the Coding a Custom DLL Task Function on page 2057 section for more information on the arguments.
- Create a DLL registry file for Creo Distributed Batch. Refer to the Registry File for Custom DLL Tasks on page 2059 section for more information.
- Create a TTD (Task Type Definition) file specifying the Creo Parametric TOOLKIT DLL and the custom task function to be executed by the custom task. Refer to the TTD File Format for Custom DLL Tasks on page 2059 section for more information.

Coding a Custom DLL Task Function

Creo Distributed Batch provides the following input arguments to the custom DLL task function:

Note

All the input arguments are of the widestring data type.

- `DBS_WORKING_DIRECTORY`—Specifies the full path of the working directory for the Creo Distributed Batch service. It stores all the`log`, `inf`, and output files generated during the execution of a custom task.
- `DBS_CURRENT_OBJECT_NAME`—Specifies the name of the object that will acted upon by the custom DLL task function.
- `DBS_CURRENT_OBJECT_TYPE`—Specifies the type of the object such as a 3D model, assembly, or drawing. The value of this argument will be a widestring containing an integer value of the type that matches with the values in the enumerated type `ProMdlType`.

Creo Distributed Batch supplies any additional input arguments and their values, if they are listed in a TTD file, as <USER_DATA></USER_DATA> tags within the <TKFUNC></TKFUNC> node. Each user data tag is supplied as an independent argument.

The custom DLL task function can influence the state of Creo Distributed Batch and the status of the task through one of the following output arguments. If an argument is not supplied, the current or default value is used.

Note

All the output arguments are of the widestring data type.

- DBS_CURRENT_OBJECT_NAME—Specifies the name of the generated object. This output informs Creo Distributed Batch that a model with this name should be used for other TTD entries that follow.
- DBS_CURRENT_OBJECT_TYPE—Specifies the type of the generated object such as a 3D model, assembly, or drawing. The value of this argument will be a widestring containing an integer value of the type that matches with the values in the enumerated type ProMdlType. This output informs Creo Distributed Batch that a model of this type should be used for other TTD entries that follow.
- DBS_IGNORE_FILE—By default, the Creo Distributed Batch service returns any file generated in the working directory, except for report type files such as log, inf and txt. The names of the files specified by this output argument will not be included in the output returned to the client. You can return more than one argument of this type.
- DBS_OUTPUT_FILE—Some file types such as log, inf and txt are not transferred to the Creo Distributed Batch client by default. File names specified by this output argument indicate that a particular file such as trail.txt should be passed back to the client. This file is always returned even if Creo Distributed Batch ignores it. This argument is not required for files returned by default. You can return more than one argument of this type.
- DBS_OUTPUT_DIRECTORY—Specifies the output directory containing all files except ones such as trail files or log files that are ignored by Creo Distributed Batch. This optional argument is required only if the contents of the output directory are different from the input working directory.
- DBS_MESSAGE—Specifies the user-visible message that will be printed in the log file generated by the custom DLL task function. This message is especially useful if the function returns an error.

In case of chained TTD tasks (described in the TTD File Format for Custom DLL Tasks on page 2059 section), the output arguments generated by the first TTD containing the custom DLL task function are supplied as input arguments to the second TTD within the same file.

The custom DLL task function should return one of the following types of errors:

- `PRO_TK_NO_ERROR`—Specifies that the task function executed successfully.
- `PRO_TK_BAD_INPUTS`—Specifies that the task function was called with incorrect input arguments.

Registry File for Custom DLL Tasks

Registering a custom Creo Parametric TOOLKIT DLL means providing information to the Creo Distributed Batch service about the files that form the DLL.

An example of the registry file is as follows:

```
name pt_userguide.dll
exec_file <creo_toolkit_loadpoint>/$<machine_type>/obj/pt_userguide.dll
text_dir <creo_toolkit_loadpoint>/protk_appls/pt_userguide/text
where:
<creo_toolkit_loadpoint> refers to the directory that forms the loadpoint of
Creo Parametric TOOLKIT under the Creo Parametric installation.
<machine_type> refers to the type of machine on which Creo Parametric
is installed. For example x86e_win64.
```

The fields of the above registry file are described below:

- `name`—Specifies the name of the Creo Parametric TOOLKIT DLL to be used as the `dllname` attribute in the TTD file.
- `exec_file`—Specifies the full path to the DLL binary file on the service machine.
- `text_dir`—Specifies the directory containing language-specific directories that include the menu and message files used by the DLL.

The registry file must be named `dbatchs.dat` The DLL registry file must be visible to the Creo Distributed Batch service on the service machine. To make it visible, place the `dbatchs.dat` file in the same location as the Creo Distributed Batch service executable `dbatchs.exe` available in the *<creo_loadpoint>**<datecode>**Common Files*\$*<machine_type>*\`nms` directory. More than one custom DLL can be listed in a given `dbatchs.dat` file.

TTD File Format for Custom DLL Tasks

TTD files serve as templates for all the common tasks that can be performed using Creo Distributed Batch. Refer to the Creo Distributed Batch Help for complete information about the nodes included in a TTD file.

For a custom DLL task, specify the name of the Creo Parametric TOOLKIT DLL as the dllname attribute and the function ProToolkitTaskExecute() that causes Creo Parametric to execute the task function as the func attribute within the <TKFUNC></TKFUNC> node in a TTD file. Specify the name of the custom DLL task function in the <DLL_FUNCTION></DLL_FUNCTION> tag within the <TKFUNC></TKFUNC> node. You can also include the required <USER_DATA></USER_DATA> tags within the <TKFUNC></TKFUNC> node.

Additionally, you can specify an existing Creo Parametric TOOLKIT function along with its input arguments and enumerated types within a <TKFUNC></TKFUNC> node. A TTD file containing multiple <TKFUNC></TKFUNC> nodes is called a chained TTD.

Example 2: Chained TTD Task to convert a Creo Parametric model into a simplified representation in the VRML format

Following is an example of a TTD file for a custom DLL task function.

```
<TTD version="1.0" created_by="PTC">
    <DESCRIPTION>Load and call external DLL to change the
 current model to simprep</DESCRIPTION>
    <DETAILS>Load an external Creo Parametric TOOLKIT DLL into the
 Creo Parametric session</DETAILS>
    <SERVICE name="dbatchs"/>
    <TKFUNC func="ProToolkitTaskExecute"
              dllname="pt_userguide.dll">
      <DLL_FUNCTION>TestSimprepActivateTask</DLL_FUNCTION>
      <USER_DATA name="SIMP_REP">NO_B</USER_DATA>
    </TKFUNC>
    <!-- Export to VRML format -->
    <TKFUNC func="ProExportVRML">
    </TKFUNC>
</TTD>
```

In the above simprep.ttd file, the function ProToolkitTaskExecute() executes the custom task function TestSimrepActivateTask defined in pt_userguide.dll. The function TestSimrepActivateTask converts a Creo Parametric model into a simplified representation. The name of the simplified representation to be created must be specified in the <USER_DATA></USER_DATA> tag.

The activated simplified representation is converted into the VRML format by the second <TKFUNC></TKFUNC> node entry in the chained TTD task. This part of the execution is directly handled by Creo Distributed Batch.

The C language code for the custom task function TestSimrepActivateTask used in the simprep.ttd file is present in the sample code in the file UgInterfaceExport.c located at <creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_interface.

Launching Synchronous J-Link Applications

The functions described in this section allow Creo Parametric TOOLKIT API users to launch a synchronous J-Link application and call methods within it with user-specified arguments.

The ability to launch and control a J-Link application by a Creo Parametric TOOLKIT API enables:

- Code reuse through the ability to create J-Link libraries that can be called from Creo Parametric TOOLKIT or other APIs.
- Creation of platform independent library routines.

Functions Introduced:

- **ProJlinkApplicationStart()**
- **ProJlinkTaskExecute()**
- **ProJlinkApplicationStop()**

Use the function `ProJlinkApplicationStart()` to register and start a J-Link application. The input parameters of this function are similar to the fields of a registry file and are as follows:

- *app_name*—Assigns a unique name to this J-Link application.
- *java_app_class*—Specifies the fully qualified class and package name of the Java class that contains the J-Link application's start and stop method.
- *java_app_start*—Specifies the start method of the program.
- *java_app_stop*—Specifies the stop method of the program.
- *java_app_add_classpath*—Specifies the locations of packages and classes that can be loaded when running a Java program. Can be `NULL`, if not needed
- *text_dir* —Specifies the application text path for menus and messages. Can be `NULL` if the application does not use menus or messages.
- *user_display*—Specifies whether to display the application in the **Auxiliary Applications** dialog box in Creo Parametric.

The function provides the `ProJlinkAppHandle` handle to the J-Link application. If the start method throws an exception, the description of the exception is stored in the argument *startup_exception*.

The function `ProJlinkTaskExecute()` calls a registered task in a J-Link application. The input parameters are:

- *handle*—Specifies the handle to the J-Link application.
- *task_id*—Specifies the task to be executed. The J-Link application should register the task using the J-Link method

`pfcSession.BaseSession.RegisterTask()`. Refer to J-Link documentation for more information on the RegisterTask() method.

- *input_args*—Specifies the input arguments to be passed to the task.

The output of this function is an array of arguments of type `ProArgument`. These arguments are returned by the J-Link task method.

If the method throws an exception, the description of the exception is stored in the output argument *exception*.

The function `ProJlinkApplicationStop()` stops the J-Link application specified by the `ProJlinkAppHandle` application handle. The function activates the application's stop method.

If the stop method throws an exception, the description of the exception is stored in the output argument *exception*.

78

Technical Summary of Changes

Technical Summary of Changes for Creo 8.0.0.0

The critical and miscellaneous technical changes in Creo Parametric 8.0.0.0 and Creo Parametric TOOLKIT are explained in this section. It also lists the new and superseded functions for this release.

Critical Technical Changes

This section describes the changes in Creo Parametric 8.0.0.0 and Creo Parametric TOOLKIT that might require alteration of existing Creo Parametric TOOLKIT, Creo Elements/Pro TOOLKIT, and Pro/TOOLKIT applications.

Secure Ports in Creo Parametric TOOLKIT

For enhanced security purpose, starting with Creo Parametric 8.0.0.0 and later, Creo Parametric TOOLKIT connection can only happen from the same machine where `xtop` is running.

Separate Download Option for ICU DLLs

For enhanced security purpose, in Creo Parametric 8.0.0.0 and later, the DLLs `ucore46.dll` and `udata46.dll` located at `creo_loadpoint>\<datecode>\Common Files\<platform>\lib` and `creo_loadpoint>\<datecode>\Common Files\<platform>\obj` are not installed by default. However, if you need to load applications that are compiled prior to the Creo 7.0 release, you must install the DLLs separately. To download and install these DLLs separately, ensure that the **Legacy Toolkit Application Runtime** checkbox is selected while installing Creo Parametric.

Compiling and Linking on Windows

In Creo Parametric 8.0.0.0 and later, Creo Parametric TOOLKIT supports Visual Studio 2019. The compiler flags and libraries are available for Visual Studio 2019. Creo Parametric TOOLKIT no longer supports Visual Studio 2015.

All Creo Parametric TOOLKIT applications on 64-bit Windows platforms built using the Microsoft Visual Studio 2019 compiler must set the configuration property Platform Toolset as Visual Studio 2019 (v142).

Updates to Error Types

Following is the list of the error types added to functions:

- ProSetdatumtagReferencesAdd(), ProNoteReferencesAdd(), and ProGtolReferencesAdd() —The error types PRO_TK_MAX_LIMIT_REACHED and PRO_TK_CANT_MODIFY are added.
  - PRO_TK_MAX_LIMIT_REACHED—indicates that ProAnnotationReference has more than 1 reference from the type PRO_ANNOT_REF_SRF_COLLECTION, and only one is allowed. If this error is returned no reference was added at all.
  - PRO_TK_CANT_MODIFY—indicates that reference from the type PRO_ANNOT_REF_SRF_COLLECTION already exists.
- ProDimensionAdditionalRefsAdd()—The error types PRO_TK_MAX_LIMIT_REACHED, PRO_TK_CANT_MODIFY and PRO_TK_BAD_CONTEXT are added.
  - PRO_TK_MAX_LIMIT_REACHED—indicates that ProDimensionReferenceType has more than 1 reference from the type PRO_DIM_SRF_COLL, and only one is allowed. If this error is returned no reference was added at all.
  - PRO_TK_CANT_MODIFY— indicates that reference from the type PRO_ANNOT_REF_SRF_COLLECTION already exists.
  - PRO_TK_BAD_CONTEXT— indicates that the reference type that is added is inconsistent with the reference type that is supported. For example, reference type PRO_ANNOT_REF_SINGLE and PRO_ANNOT_REF_SRF_COLLECTION.
- ProAsmcompAssemble()—The error type PRO_TK_GENERAL_ERROR is added. It indicates that the component creation failed. The function ProAsmcompAssemble() returns this error if it is called to add an embedded component in a different owner assembly.
- ProFeatureWithoptionsCreate()—The error type PRO_TK_UNSUPPORTED is added. It indicates that the embedded component feature is created in a different owner assembly or subassembly.
- ProFeatureWithoptionsRedefine()—The error type PRO_TK_UNSUPPORTED is added. It indicates that the embedded component feature is redefined in a different owner assembly or subassembly.
- ProAsmcompFillFromMdl()—The error type PRO_TK_UNSUPPORTED is added. It indicates that the model to which the template model is copied is not supported. For example, it is an embedded model.
- ProDrawingFromTemplateCreate()—The error types PRO_TK_INVALID_NAME and PRO_TK_DWGCREATE_ERRORS are added.
  - PRO_TK_INVALID_NAME— indicates that the template and/ or the model name of the drawing is an embedded model name.

- PRO_TK_DWGCREATE_ERRORS—indicates that there are one or more errors while creating the drawing.

New Functions

This section describes new functions for Creo Parametric TOOLKIT for Creo Parametric 8.0.0.0.

Annotations

New Function	Description
ProNoteURLExtraInfoGet()	Retrieves the information of whether opening the URL for a specified note appends the extra information "?<model name>+<note id>".
ProNoteURLExtraInfoSet()	Sets whether opening the URL for a specified note should append the extra information "?<model name>+<note id>".

Assemblies and Components

New Function	Description
ProAsmcompEmbed()	Embeds selected components in its owner assembly.
ProAsmcompExtract()	Extracts the embedded component from the owner assembly.
ProAsmcompEmbeddedOwnerMdlGet()	Returns the handle of the nonembedded owner model for the specified embedded model.

Cabling

New Function	Description
ProConnectorRefModelNameGet()	Retrieves the reference model name of the specified cable connector.

Cross Section

New Function	Description
ProXSectionCreateDataAlloc()	Allocates memory for the `ProXSectionCreateData` data structure.
ProXSectionCreateDataFree()	Releases the memory of the `ProXSectionCreateData` data structure .
ProXSectionCreateDataQuiltSelGet() ProXSectionCreateDataQuiltSelSet()	Gets and sets the quilt selection data.
ProXSectionCreateDataQuiltTypeGet() ProXSectionCreateDataQuiltTypeSet()	Gets and sets the quilt cross section type using the structure `ProXSectionCreateData`.

Data Exchange

New Function	Description
ProIntfExportProfileLoad()	Loads the specified profile for export.

Drawing

New Function	Description
ProDrawingDraftViewsCollect()	Collects all draft views in the specified drawing.
ProDrawingViewIsDraft()	Determines whether the specified view is a draft view.
ProDrawingDraftViewCreate()	Creates a draft view in the specified drawing sheet.
ProDrawingDimAttachpointsViewGet()	Retrieves the attachments and sense of the specified drawing dimension. This function fetches and interprets the attachment in the context of the view in which the dimension is placed.

Features

New Function	Description
ProFeatureReferenceEditRefsGet()	Returns an array of the original references of a feature that are used to perform the edit reference operation.

Fundamentals

New Function	Description
ProToolkitMajorVersionGet()	Returns the version number of the Creo Parametric executable to which the Creo Parametric TOOLKIT application is connected.

Graphics

New Function	Description
ProTextFontRetrieve()	Loads a font with the specified name that can be used to display the text.
ProMatrixMakeOrthonormal()	Converts a non-orthonormal matrix to an orthonormal matrix with the specified scaling factor.

Models

New Function	Description
ProMdlIsEmbeddedName()	Checks if the specified model name or full path that includes the model name is an embedded model name.
ProMdlVisibleGet()	Returns the handle to the generic or visible model for the specified model.

Production Applications: Welding

New Function	Description
ProWeldExtendedInfoToXMLExport()	Prints the information that is necessary to automatize the welding info file, in XML format.

Relations

New Function	Description
ProRelationEvalWithUnitsRefResolve()	Evaluates the expression that is specified on the right side of a relation line and returns the value in the form of `ProParamvalue` structure.

Simplified Representations

New Function	Description
ProAutomaticSimprepRetrieve()	Retrieves a user-defined simplified representation as automatic representation.
ProAutomaticSimprepConvert()	Converts a user-defined representation to automatic simplified representation while maintaining the excluded or substituted components in the representation.
ProAutomaticSimprepActivate()	Activates a user-defined representation as an automatic simplified representation.

Solids and Parts

New Function	Description
ProSolidMaxsizeGet()	Retrieves the maximum model size of the specified solid.
ProAssemblySolidMassPropertyGet()	Calculates the mass properties of a solid that is referenced by the specified coordinate system selection.

Surface Properties

New Function	Description
ProSurfaceAppearanceDefaultPropsGet()	Gets the default appearance properties of the specified type of surface.

Symbol Instance

New Function	Description
ProDtlsyminstReferencesAdd()	Adds semantic references to a specified symbol.
ProDtlsyminstReferencesGet()	Returns a `ProArray` of additional semantic references for a symbol.
ProDtlsyminstReferenceDelete()	Deletes the additional semantic references.

User Interface: Dashboards

New Function	Description
ProUIDashboardshowoptionsDefaultOpenSet()	Sets the specified dashboard as the open by default page.
ProUIDashboardStdlayoutDefaultBtnsAdd()	Adds new standard push buttons to the Creo Parametric dashboard.
ProUIDashboardStdlayoutButtonAdd()	Adds a new push button to the Creo Parametric dashboard.
ProUIDashboardStdlayoutDefaultButtonNameGet()	Returns the default name of the specified button id.
ProUIDashboardPauseresumeButtonStateGet() ProUIDashboardPauseresumeButtonStateSet()	Returns and sets the pause and resume state of the button.
ProUIDashboardpageStateSet()	Modifies the visibility of the button that opens the dashboard page according to the page state.

User Interface: Dialogs

New Function	Description
ProUIPushbuttonModaloverrideSet()	Sets the `pushbutton` modal override for the specified dialog and component according to the value defined by the enumerated data type `ProUIModalOverride`.
ProUICheckbuttonModaloverrideSet()	Sets the `checkbutton` modal override for the specified dialog and component according to the value defined by the enumerated data type `ProUIModalOverride`.
ProUIDialogAppActionSet()	Sets a function to be called only once, when you return to or enter an event loop.
ProUIDialogAppActionRemove()	Removes a function added via `ProUIDialogAppActionSet()`.

Superseded Functions

This section describes the superseded functions for Creo Parametric TOOLKIT for Creo Parametric 8.0.0.0.

Fundamentals

Superseded Function	New Function
ProEngineerReleaseNumericversionGet()	ProToolkitMajorVersionGet()

Relations

Superseded Function	New Function
ProRelationEvalWithUnits()	ProRelationEvalWithUnitsRefResolve()

Miscellaneous Technical Changes

The following changes in Creo Parametric 8.0.0.0 can affect the functional behavior of Creo Parametric TOOLKIT. PTC does not anticipate that these changes cause critical issues with existing Creo Parametric TOOLKIT, Creo Elements/Pro TOOLKIT, or Pro/TOOLKIT applications.

Ability to Create Multiple Holes using Sketch Points as Hole Placement

In Creo Parametric 8.0.0.0, the elements `PRO_E_STD_SECTION` and `PRO_E_HOLE_SKDP_OPTIONS` are added in the element tree for the Hole Placements in the header file `ProHole.h`. These elements enable you to create multiple holes in a single hole feature. The holes can be placed on sketched entities such as sketched points, end-points or mid-points of sketched lines.

Event Callback function ProFeatureNeedsRegenGet is Deprecated

The event callback function `ProFeatureNeedsRegenGet()` is executed multiple times for the type `PRO_FEATURE_NEEDS_REGEN_GET` for the function `ProNotificationSet()`. Preferably, the number of calls made to the callback function must be equivalent to the number of features. However, the callback function is getting called multiple number of times. Hence, the event callback function is deprecated and does have a successor function added.

Support for Miter Cuts in Flat Walls

In Creo Parametric 8.0.0.0 and later, the following elements are added to the for element tree for Flat Wall feature in the `ProSmtFlatWall.h` header file:

- `PRO_E_SMT_MTR_CUTS_ADD`—Specifies the miter cuts to be added.
- `PRO_E_SMT_THREE_BEND_CRNR_RELIEF_TYPE`—Specifies the three bend corner relief type and is defined by the enumerated data type `ProThreeBendCornerType`.
- `PRO_E_SMT_MITER_CUT_GROOVE_TYPE`—Specifies the groove type to be cut in the miter and is defined by the enumerated data type `ProMiterCutType`.
- `PRO_E_SMT_MTR_CUTS_WIDTH_VAL`—Specifies the width value of the miter cut.
- `PRO_E_SMT_MTR_CUTS_OFFSET_VAL`—Specifies the offset value of the miter cut.

Support for Standard Tapered Holes

In Creo Parametric 8.0.0.0 and later, the element tree for `ProHole.h` is updated with standard tapered hole feature elements to control the depth and diameter of the tapered hole. You can now use straight drill for a tapered hole. The following elements have been added:

- `PRO_E_HLE_TAPERED_STRT_DEPTH_OPT`—You can specify the depth type of the straight drill using the enumerated data type `ProHleTaperStrDepType`
- `PRO_E_HLE_ADD_TAPERED_TIP_ANGLE`—You can choose to add or not add the tapered tip angle in the tapered hole using the enumerated data type `ProHleAddTaperedTipAngFlag`. A tapered tip is a slanted surface at the bottom of the tapered drill.
- `PRO_E_HLE_TAPERED_STRT_DIA`—You can specify the diameter of the tapered straight hole.
- `PRO_E_HLE_TAPERED_STRT_DEPTH`—You can specify the depth of the tapered straight hole. It depends on the element `PRO_E_HLE_TAPERED_STRT_DEPTH_OPT`. The tapered hole option is available only when the blind depth value is specified using the enumerated date type `ProHleTaperStrDepType`.
- `PRO_E_HLE_TAPERED_TIP_ANGLE`—You can specify the tapered tip angle. It depends on the element `PRO_E_HLE_ADD_TAPERED_TIP_ANGLE`. The tapered tip angle option is available only when the add tapered tip value is specified using the enumerated data type `ProHleAddTaperedTipAngFlag`

Creating a Draft Feature Using Round Surfaces

The PRO_DRAFT_UI_RND_HINGE value is added to the enumerated data type ProDraftHingeType in the ProDraft.h header file. When you create a draft feature you can now select a round surface as a type of draft hinge. This round surface must be adjacent to the draft surface.

Support for Tape over Multiple Branches

The PRO_CABLECOSMTYPE_BRANCH_TAPE value is added in the enumerated data type ProCableCosmeticType in the ProCabling.h header file. This new value enables you to collect information about the new cosmetic branch tape feature. You can apply this new feature at branch points of the harnesses where the tape wraps around each cable branch.

Creating a Datum Plane Using a Datum Point or a Vertex

The PRO_DTMPLN_FIT_POINT value is added in the enumerated data type ProDtmplnFitType in the ProDtmPln.h header file. When you create a datum plane, as a size reference, you can select a datum point or a vertex as the center point. This ensures that the datum plane boundaries are centered around the reference that is used to create a datum plane. To set the datum plane outline width and height, set the values of the configuration options datum_outline_default_width and datum_outline_default_height, respectively.

Elements for Bend Relief Length Added

The elements PRO_E_BEND_RELIEF_LENGTH_TYPE and PRO_E_BEND_RELIEF_LENGTH are added to the following element trees:

- ProSmtBend.h
- ProSmtEdgeBend.h
- ProSmtEditBendRelief.h
- ProSmtFlatWall.h
- ProSmtJoinWalls.h

The element PRO_E_BEND_RELIEF_LENGTH_TYPE specifies the type of the relief length and the element PRO_E_BEND_RELIEF_LENGTH specifies the value of relief length.

ProReferenceTypeGet() returns ProType for Specific Dimension Attachment Reference

When a surface has multiple silhouettes, the function ProReferenceTypeGet() returns a value between PRO_SILH_EDGE and PRO_SILH_EDGE_MAX. The silhouette index is given by Protype value minus PRO_SILH_EDGE.

Support for Datum Axis Types that are Parallel and Normal to a Linear Entity

When you create a feature, you can use the options `PRO_DTMAXIS_PARALLEL` and `PRO_DTMAXIS_NORM_ENT` added in the enumerated data type `ProDtmAxisType` in the `ProDtmAxis.h` header file. These two options allow you to create datum axes parallel and normal to linear entity respectively, and can be used as references for feature creation.

- `PRO_DTMAXIS_PARALLEL`—Specify this option when you want to create a datum axis parallel to the linear entity and that passes through a datum point or vertex
- `PRO_DTMAXIS_NORM_ENT`—Specify this option when you want to create a datum axis that is normal to a linear entity and that passes through a datum point or vertex

Support for Server Registration Error Messages

While performing server registration, the following errors might be thrown:

- `PRO_TK_BROWSER_UNAVAILABLE`—When the browser service is unavailable and fails to initialize.
- `PRO_TK_DLL_LOAD_ERROR`—When the file `prowt.dll` does not load.

In both the situations, you need to verify the integrity of your installation and try registering the server again.

The above error messages are added to the header file `ProToolkitErrors.h`.

Support for Redefining a Combined State with Most Recently Used Reference States

If you want to create or redefine a combined state, which when invoked leaves some reference states unchanged, as achieved in the UI for the combined states via the 'Most Recently Used' option, use the value `PRO_COMBSTATE_REF_MRU` as the id for that type of reference state.

Full Version of Creo® Parametric TOOLKIT Release Notes

To see a full version of the *Creo® Parametric TOOLKIT Release Notes*, visit the page Creo® Parametric TOOLKIT Release Notes. The full version contains information from all the past release notes for Creo Parametric TOOLKIT.

Technical Summary of Changes for Creo 8.0.1.0

The critical and miscellaneous technical changes in Creo Parametric 8.0.1.0 and Creo Parametric TOOLKIT are explained in this section. It also lists the new and superseded functions for this release.

New Functions

This section describes new functions for Creo Parametric TOOLKIT for Creo Parametric 8.0.1.0.

Annotations: Features

New Function	Description
ProAnnotationSecuritymarkingSet()	Sets the security marking option for notes and symbols.
ProAnnotationSecuritymarkingGet()	Retrieves the security marking option for notes and symbols.

Drawings

New Function	Description
ProDtlentitydataIsPeriodic()	Checks if the draft identity is marked as periodic.
ProDtlentitydataPeriodicSet()	Marks the draft entity to be periodic.

Features

New Function	Description
ProFeatureMdltreeDisplaynameGet()	Returns the name of the nodes in the model tree.

Full Version of Creo® Parametric TOOLKIT Release Notes

To see a full version of the *Creo® Parametric TOOLKIT Release Notes*, visit the page Creo® Parametric TOOLKIT Release Notes. The full version contains information from all the past release notes for Creo Parametric TOOLKIT.

Technical Summary of Changes for Creo 8.0.2.0

The critical and miscellaneous technical changes in Creo Parametric 8.0.2.0 and Creo Parametric TOOLKIT are explained in this section. It also lists the new and superseded functions for this release.

New Functions

This section describes new functions for Creo Parametric TOOLKIT for Creo Parametric 8.0.2.0.

Assembly

New Function	Description
ProExpldStateExplodeLinesGet()	Returns an array of explode lines for the specified exploded state.

Cross-Sections

New Function	Description
ProOffsetXsecInfoGet()	Returns the parameters for a specified offset cross section.

Full Version of Creo® Parametric TOOLKIT Release Notes

To see a full version of the *Creo® Parametric TOOLKIT Release Notes*, visit the page Creo® Parametric TOOLKIT Release Notes. The full version contains information from all the past release notes for Creo Parametric TOOLKIT.

A

Unicode Encoding

This appendix describes how the new Unicode support used internally by Pro/ENGINEER from Wildfire 4.0 onward affects Creo Parametric TOOLKIT and its applications.

Introduction to Unicode Encoding

UNICODE is an acronym for "Universal Character Encoded System". It is a unique character encoding scheme allowing characters from European, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, Thai, Urdu, Hindi, and other world languages to be encoded in a single character set. This enables applications to simultaneously support text in multiple languages in their data files. Unicode encoding covers most of the letters, punctuation marks, and technical symbols commonly used in the English language that are not covered by the legacy encoding.

Unicode defines two mapping methods:

- `UCS (Universal Character Set)` encoding
- `UTF (Unicode Transformation Format)` encoding

For more information on Unicode Encoding, visit http://unicode.org.

Pro/ENGINEER Wildfire 4.0 onward, all string data in Pro/ENGINEER (previously stored in the legacy encoding format) is now stored in the Unicode encoding. Pro/ENGINEER Wildfire 4.0 uses the `UCS-2` encoding on Windows platforms and `UCS-4` encoding in UNIX environments for widestring data. It reads and writes character data using the mulitbyte `UTF-8` encoding on all platforms. `UTF-8` is an 8-bit, variable-length character encoding format that uses one to four bytes per character.

Some important terminology about string encoding related to Creo Parametric TOOLKIT that is used throughout this appendix is described as follows:

- “Unicode encoding” refers to the string and widestring encodings used by Pro/ENGINEER Wildfire 4.0 and later.
- “Legacy encoding” refers to the encoding used by Pro/ENGINEER Wildfire 3.0 and earlier. Depending on the language, this encoding is typically some version of an `EUC` encoding.
- “Native encoding” refers to the encoding used by the operating system in the language in which the system is running. This encoding is the same as legacy encoding in most cases.
- “Multibyte string” refers to a character array representing a string in the C language. Because of the limited size of the character (a single byte), combinations of multiple bytes are used to represent characters outside the ASCII range.
- “7-bit ASCII” refers to the character range 0x0 through 0x127. This range is shared between Unicode and non-Unicode encodings used by Creo Parametric. Thus, any data of this type is unchanged after transcoding.
- “8-bit ASCII” refers to the character range 0x128 through 0x255. In many European native encodings, this range is used to represent European accented

vowels and other letters. In Unicode, this range is not directly used. Therefore, 8-bit ASCII native strings are not equivalent in Unicode.

- "Byte Order Mark" (BOM) refers to a string of three bytes `U+FEFF` (represented in C language strings by "\357\273\277"), and is placed on the top of a text file to indicate that the text is Unicode encoded. Unicode has designated the character `U+FEFF` as the BOM and reserved `U+FFFE` as an illegal character for `UTF-8` encoding. Most of the text files generated by Creo Parametric are written with the BOM and Unicode encoding. Creo Parametric can accept a Unicode encoded text file with a BOM, or a legacy encoded text file without a BOM as the input.
- "Transcoding" refers to the act of changing a string or widestring encoding from one encoding to another, for example, from platform native to Unicode or vice-versa. For some transcoding operations, there is a possibility of data loss, since characters from one encoding may not be supported in the target encoding.

Unicode Encoding and Creo Parametric TOOLKIT

Pro/TOOKIT applications running with Pro/ENGINEER Wildfire 4.0 and later must, by default, receive and send strings and widestrings to Pro/ENGINEER in Unicode encoding. This is a change to the encoding previously received by applications in Wildfire 3.0 and earlier. Because the workstation operating system will not be running in Unicode and other languages, functions and libraries accessed by the Creo Parametric TOOLKIT application may not be Unicode aware, the Creo Parametric TOOLKIT application must deal with the change of encoding.

Make changes to the application to expect and accept Unicode strings and widestrings when dealing with Creo Parametric data. At the external interfaces from the application to the operating system or third-party APIs, perform necessary transcoding operations to ensure that those other systems receive an expected encoding.

PTC recommends that all applications be evaluated for Unicode compliance regardless of their purpose or intended data. However, applications that would particularly be affected by Unicode encoding are as follows:

- Any Creo Parametric TOOLKIT application expected to work with Creo Parametric in any language other than English.
- Any Creo Parametric TOOLKIT application expecting Creo Parametric data in any language other than English (where strings from that data are transferred to and from Creo Parametric or any other source).

Necessity of Unicode Compliance

It is strongly recommended that you make your existing Creo Parametric TOOLKIT applications Unicode-compliant for the following reasons:

- Applications that are not Unicode-compliant will be unable to reliably handle Creo Parametric data saved in the Unicode format with strings (notes, annotations, table, and so on) in multiple languages other than English. For example, a Creo Parametric drawing can now contain both German and Japanese notes. The Creo Parametric TOOLKIT application will not be able to read or modify those notes correctly without being Unicode compliant. This could result in data loss or corruption.
- Applications that do not consider the Unicode nature of Creo Parametric data may try to pass that data directly to the system or third-party APIs that do not recognize it correctly. This could cause data corruption or crashes.
- Applications that do not transcode non-Unicode data into Unicode before using the data as strings inside Creo Parametric models will generate corrupt and incorrect models.

External Interface Handling

Creo Parametric TOOLKIT applications running in Unicode will need to create utilities around the interfaces between non-Unicode aware third-party APIs and interfaces. While PTC cannot directly provide such interfaces, this section discusses the considerations for creating such utilities by showing how one external API such as the C runtime library can be used from a Unicode environment.

Any C runtime library accepting `char*` or `wchar_t*` as input may be adversely affected by receiving Unicode data. Typically, it should be possible to create a simple wrapper for each C runtime interface used in the application, where the input string to the interface is expected to be in Unicode. The string should be transcoded before calling the system API. Examples of such C runtime functions are listed below (this is not an exhaustive list):

- fopen()
- access(), _access()
- chdir(), readdir(), opendir()
- chmod(), _chmod()
- findfirst(), _findnext()
- getcwd()
- getenv()

- open(), opendir()
  - fgetc()
  - fgets()
  - fputc()
  - fputs()
  - fread(), fwrite()
  - puts()
- remove(), stat(), system(), tmpfile(), unlink()

Special External Interface: printf() and scanf() Functions

The `printf()` and `scanf()` family of C runtime functions are a special case of an external interface. The format string must be transcoded when these interfaces are called. The list of variable arguments passed to the functions may also contain string and widestring data that needs to be transcoded and modified in format. Because of the complexity of wrapping these C runtime functions, PTC has provided a standard Creo Parametric TOOLKIT function equivalent for each. These functions support all the format specifiers and modifiers supported by the C language specification.

Functions Introduced:

- **ProTKPrintf()**
- **ProTKFprintf()**
- **ProTKSprintf()**
- **ProTKSnprintf()**
- **ProTKVprintf()**
- **ProTKVfprintf()**
- **ProTKVsprintf()**
- **ProTKVsnprintf()**
- **ProTKScanf()**
- **ProTKFscanf()**
- **ProTKSscanf()**
- **ProTKSnscanf()**
- **ProTKVscanf()**
- **ProTKVfscanf()**
- **ProTKVsscanf()**

The function `ProTKPrintf()` provides the Unicode equivalent to the C runtime function `printf()`. The number of characters returned by this function is sent to stdout. The output data is transcoded to the native encoding format, which may result in out-of-locale characters in the results.

The function `ProTKFprintf()` provides the Unicode equivalent to the C runtime function `fprintf()`. The number of characters returned by this function are copied into the file. This file will receive the data in the Unicode-encoded format.

The functions `ProTKSprintf()` and `ProTKSnprintf()` provide the Unicode equivalent to the C runtime functions `sprintf()` and `snprintf()` respectively. The number of characters returned by these functions are copied into the output buffer.

The function `ProTKScanf()` provides the Unicode equivalent to the C runtime function `scanf()`. This function parses the contents of the input from stdin. The output data in the string or character format is in Unicode encoding.

The function `ProTKFscanf()` provides the Unicode equivalent to the C runtime function `fscanf()`. This function parses the contents of the input from a file.

The functions `ProTKSscanf()` and `ProTKSnscanf()` provide the Unicode equivalent to the C runtime functions `sscanf()` and `snscanf()` respectively.

The Unicode equivalent of the C runtime functions `v*printf()` and `v*scanf()`, which take a variable arguments list instead of variable number of arguments, have also been provided in the form of `ProTKV*printf()` and `ProTKV*scanf()` functions.

Special External Interface: Windows-runtime Functions

Win32 functions that take char* inputs are not Unicode compliant, and thus cannot be used with data directly obtained from Pro/ENGINEER Wildfire 4.0 and later. The simplest approach to using Windows runtime functions is to use the functions accepting wchar_t* inputs since these functions are Unicode compliant (Windows native encoding for whar_t* is Unicode). For example, use the function `GetMessageW()` instead of `GetMessage()` or `GetMessageA()`.

Special External Interface: Hardcoded Strings

Another example of an external interface is a hardcoded string. You should review all uses of hardcoded strings in your application and ensure that they fit the following categories:

- They use only 7-bit ASCII characters or wide characters.
- They use Unicode escape sequences.

8-bit ASCII or non-Unicode escape sequences in hardcoded strings do not work correctly unless you transcode the string into Unicode before sending it to Creo Parametric.

Example 3: Write a Unicode-encoded widestring from Creo Parametric to a file containing BOM

The sample code in the file `UgUnicodeTranscoding.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main` writes a unicode-encoded widestring from Creo Parametric to a file containing BOM.

Example 4: Write Unicode-encoded widestring to Creo Parametric after reading the unicode string from a file containing the BOM

The sample code in the file `UgUnicodeTranscoding.c` located at `<creo_toolkit_loadpoint>/protk_appls/pt_userguide/ptu_main` writes a unicode-encoded widestring to Creo Parametric after reading the unicode string from a file containing the BOM.

B

Updating Older Applications

This appendix describes the tools that are available for updating applications from older versions to the current release of Creo Parametric TOOLKIT.

Overview

Creo Parametric TOOLKIT users are responsible for updating their applications from older versions to use the new Creo Parametric TOOLKIT functions. When you update the applications, use the functions that have been updated in the current release.

PTC provides tools that facilitate updating of your Creo Parametric TOOLKIT applications from older versions. However, in certain scenarios, the tools may not give expected results.

Tools Available for Updating Applications

PTC provides the `mark_deprecated.pl` tool to update your applications from older versions to the current release. This perl script is located at `<creo_loadpoint>\<datecode>\Common Files\protoolkit\scripts`. The script parses directories and looks for files with extensions such as, `.c`, `.cxx`, and `.cpp`, which are the default extensions included in the script.

The script searches for deprecated symbols, such as, functions, structures, enumerated data types and so on, in these files. When it finds such symbols, the script inserts comments with recommendations of possible replacements. By default, the script uses the mapping table `protkmap.txt` provided at `<creo_loadpoint>\<datecode>\Common Files\protoolkit\scripts`.

The mapping table is based on DEPRECATED and SUCCESSORS tags specified in the Creo Parametric TOOLKIT header files. Some deprecated functions may not have direct replacement functions. In such cases, the script adds the comment AS APPROPRIATE.

The script options are listed when you run `mark_deprecated.pl` with no arguments, or with options `-h`, `-?` or `-help`.

Note

The options may vary in future releases.

When you run the script, it saves a copy of the original files. However, PTC recommends that you back up all the files, before running the script.

The script generates two types of output. The outputs depend on the `-m` option.

For example, consider a file with the name `model.c`. It contains the following code:

```
void findMdlParam(ProMdl mdl) {
  ProMdlnameShortdata *modelList = NULL;
/* should we call ProMdlInit here?? */
  error = ProMdlDependenciesList (mdl, &modelList, &noOFModels);
  error = ProMdlCopy (mdl); // what to do?
```

```
/*who knows? That's a question
  btkString errorStr = "\"ProMdlCopy (mdl)\" - error";
  PrintModelDependencies(modelList);
  /*
   * This is long multiline comment, where some Pro/TK
   * calls like ProCollectionAlloc or ProCollectioninstrAlloc
   * can be found.
   */
  return;
}
```

If you specify the `-m` option, the original file is retained, and the output file with the name `<filename>_markup.c` is created. The comments recommending new symbols are inserted on separate lines in the output file. For the file `model.c`, the following output is generated in the file `model_markup.c`:

```
void findMdlParam(ProMdl mdl) {
  ProMdlnameShortdata *modelList = NULL;
/* should we call ProMdlInit here?? */
/* Deprecated_API_Used: Replace ProMdlInit with ProMdlnameInit */
  error = ProMdlDependenciesList (mdl, &modelList, &noOfModels);
/* Deprecated_API_Used: Replace ProMdlDependenciesList
with ProMdlDependenciesMdlnameList */
  error = ProMdlCopy (mdl); // what to do? /*who knows? That's a question
/* Deprecated_API_Used: Replace ProMdlCopy with ProMdlnameCopy */
  btkString errorStr = "\"ProMdlCopy (mdl)\" - error";
  PrintModelDependencies(modelList);
  /*
   * This is long multiline comment, where I can also note some Pro/TK
   * calls like ProCollectionAlloc or ProCollectioninstrAlloc
** Deprecated_API_Used: Replace ProCollectionAlloc with
ProCrvcollectionAlloc, ProSrfcollectionAlloc **
** Deprecated_API_Used: Replace ProCollectioninstrAlloc with AS APPROPRIATE **
   * can be found.
   */
  return;
}
```

If you do not specify the `-m` option, a copy of the original file is saved as `<filename>.c.orig`. The comments with recommendations are inserted in the original file. For example, the original file is saved as `model.c.orig`. The following output is generated in the file `model.c`:

```
void findMdlParam(ProMdl mdl) {
 ProMdlnameShortdata *modelList = NULL;
/* should we call ** PROTK_DEPRECATED ProMdlInit ->
ProMdlnameInit ** ProMdlInit here?? */
 error = /* PROTK_DEPRECATED ProMdlDependenciesList ->
```

```
ProMdlDependenciesMdlnameList */
ProMdlDependenciesList (mdl, &modelList, &noOfModels);
 error = /* PROTK_DEPRECATED ProMdlCopy ->
ProMdlnameCopy */ ProMdlCopy
(mdl);
// what to do? /*who knows? That's a question
 btkString errorStr = "\"ProMdlCopy (mdl)\" - error";
 PrintModelDependencies(modelList);
 /*
  * This is long multiline comment, where some Pro/TK
  * calls like ** PROTK_DEPRECATED ProCollectionAlloc -> ProCrvcollectionAlloc,
ProSrfcollectionAlloc ** ProCollectionAlloc or ** PROTK_DEPRECATED->
AS APPROPRIATE ** ProCollectioninstrAlloc
  * can be found.
  */
 return;
}
```

C

Migrating to Creo Object TOOLKIT C++

This appendix describes how to migrate to Creo Object TOOLKIT C++.

Overview

PTC introduced Creo Object TOOLKIT C++ as a part of its modernization plan and to improve productivity.

To enable users to smoothly migrate their existing Creo Parametric TOOLKIT applications, the Creo Parametric TOOLKIT APIWizard provides links to equivalent Creo Object TOOLKIT C++ methods.

Additionally, from Creo 3.0 onward, PTC provides a tool to facilitate migration of Creo Parametric TOOLKIT applications to Creo Object TOOLKIT C++.

Certain functional areas are not yet available in Creo Object TOOLKIT C++. In such cases, you can continue using the Creo Parametric TOOLKIT functions in Creo Object TOOLKIT C++ applications, as both the toolkits are compatible. Refer to the *Creo Object TOOLKIT C++ User's Guide* for more information.

Migrating Applications Using Tools

PTC provides the `mark_otkmethod.pl` tool to help you migrate applications from Creo Parametric TOOLKIT to Creo Object TOOLKIT C++. This perl script is located at `<creo_loadpoint>\<datecode>\Common Files\protoolkit\scripts`. The script parses directories and looks for files with extensions such as, .c, .cxx, and .cpp, which are the default extensions included in the script.

The script searches for Creo Parametric TOOLKIT functions in these files. When it finds Creo Parametric TOOLKIT functions, the script inserts comments with recommendations of possible replacing Creo Object TOOLKIT C++ methods. By default, the script uses the mapping table `protk2otkmap.txt` provided at `<creo_loadpoint>\<datecode>\Common Files\protoolkit\scripts`.

The script options are listed when you run `mark_otkmethod.pl` with no arguments, or with options `-h`, `-?` or `-help`.

Note

The options may vary in future releases.

When you run the script, it saves a copy of the original files. However, PTC recommends that you back up all the files, before running the script.

The script generates two types of outputs depending on the `-m` option. The `mark_otkmethod.pl` generates output similar to the `mark_deprecated.pl` script.

For example, consider a file `Xsection.c`. It contains the following code:

```
ProMessageDisplay(msgfil, "USER Pick the start plane");
```

```
    status = ProSelect("face", 1, NULL, NULL, NULL, NULL, &sel, &n_sel);
    if(status != PRO_TK_NO_ERROR || n_sel < 1)
        return(0);

    ProSelectionModelitemGet(sel[0], &surface_modelitem);
    ProSurfaceInit(part, surface_modelitem.id, &surface);
    ProSurfaceTypeGet(surface, &stype);
    if(stype != PRO_SRF_PLANE)
return(0);
```

If you specify the `-m` option, the original file is retained, and the output file with the name `<filename>_markup.c` is created. The recommended Creo Object TOOLKIT C++ methods are inserted as comments on separate lines in the output file. In this example, for the file `Xsection.c`, the following output is generated in the file `Xsection_markup.c`:

```
   ProMessageDisplay(msgfil, "USER Pick the start plane");
  /* Replace ProMessageDisplay with pfcSession::UIDisplayMessage /
        pfcSession::UIDisplayLocalizedMessage */

    status = ProSelect("face", 1, NULL, NULL, NULL, NULL, &sel, &n_sel);

/* Replace ProSelect with pfcBaseSession::Select */
     if(status != PRO_TK_NO_ERROR || n_sel < 1)
        return(0);

    ProSelectionModelitemGet(sel[0], &surface_modelitem);

/* Replace ProSelectionModelitemGet with pfcSelection::GetSelModel /
        pfcSelection::GetSelItem */
    ProSurfaceInit(part, surface_modelitem.id, &surface);
    ProSurfaceTypeGet(surface, &stype);
    if(stype != PRO_SRF_PLANE)
return(0);
```

If you do not specify the `-m` option, a copy of the original file is saved as `<filename>.c.orig`. The comments with recommendations are inserted in the original file. In this example, the original file `Xsection.c` is saved as `Xsection.c.orig`. The following output is generated in the file `Xsection.c`:

```
   /* Replace ProMessageDisplay -> pfcSession::UIDisplayMessage /
        pfcSession::UIDisplayLocalizedMessage */
           ProMessageDisplay(msgfil, "USER Pick the start plane");

 status =/* Replace ProSelect -> pfcBaseSession::Select */
   ProSelect("face", 1, NULL, NULL, NULL, NULL,  &sel, &n_sel);
```

```
    if(status != PRO_TK_NO_ERROR || n_sel < 1)
        return(0);

    /* Replace ProSelectionModelitemGet -> pfcSelection::GetSelModel /
        pfcSelection::GetSelItem */
          ProSelectionModelitemGet(sel[0], &surface_modelitem);

    ProSurfaceInit(part, surface_modelitem.id, &surface);
    ProSurfaceTypeGet(surface, &stype);
    if(stype != PRO_SRF_PLANE)
return(0);
```

D

Migrating to the Multibody Environment

This appendix describes how to migrate to the Creo Parametric multibody environment.

Overview

PTC introduces multibody part design to improve design productivity, flexibility, and usability.

The existing Creo Parametric TOOLKIT application continues to work seamlessly for legacy models and models having a single body. However, you must upgrade to Creo Parametric 7.0.0.0 to use the multibody environment.

To support the multibody environment, Creo Parametric 7.0.0.0 is updated in the following aspects:

- New APIs are added
- Some existing APIs are deprecated and are superseded by new APIs
- Implementation of existing APIs is updated. No visible changes in Creo Parametric Toolkit APIs
- New element trees are added to support new multibody features
- Some existing element trees are updated.
- Some enumerated data types and their values are updated
- Existing structures are updated

To enable you to smoothly migrate your existing Creo Parametric TOOLKIT applications, the Creo Parametric TOOLKIT APIWizard provides the following APIs to support multibody:

Feature Name	API name
Body Operations	• `ProSolidBodyCreate()` • `ProSolidBodyDelete()` • `ProSolidDefaultBodySet()` • `ProSolidBodyConstruc-tionSet()`
Querying Body	• `ProSolidBodiesCollect()` • `ProSolidBodySurfaceVi-sit()` • `ProSolidDefaultBodyGet()` • `ProSolidBodyStateGet()` • `ProSolidBodyIsConstruc-tion()` • `ProSolidBodyOutlineGet()` • `ProSolidBodyFeatures-Get()` • `ProGeomitemBodyGet()`

Feature Name	API name
Interference	• ProVolumeInterferenceBodiesGet() • ProVolumeInterferenceDisplayForBody()
Material Properties	• ProSolidBodyMaterialSet() • ProSolidBodyMaterialGet() • ProSolidBodyMaterialGet() • ProSolidBodyDensityGet() • ProSolidBodyMassPropertyGet()
Sheetmetal	• ProSolidBodyIsSheetmetal()
Cross-Section	• ProXSectionExcludeCompGet() • ProXSectionItemDataGet() • ProXSectionItemFree() • ProXSectionItemXhatchStyleGet() • ProXSectionItemXhatchStyleSet() • ProXSectionItemsArrFree() • ProXSectionItemsCollect() • ProXSectionOffsetCreate() • ProXSectionPlanarCreate() • ProXsecMdlnameAlloc() • ProXsecMdlnameFree() • ProXsecNewXhatchStyleCreateFromName() • ProXsectionCompXhatchStyleGet() • ProXsectionCompXhatchStyleSet()

Feature Name	API name
	• ProXsecMdlnameNameGet() • ProXsecMdlnameNameSet() • ProXsecMdlnameSolidOwnerGet() • ProXsecMdlnameSolidOwnerGet() • ProXsecMdlnameSolidOwnerSet()
Shrinkwrap options	• ProShrinkwrapoptionsIgnoreconstrbodiesSet()
UDF	• ProUdfFileIsPreCreo7()

Impact on Existing APIs

You must evaluate the need for using the new APIs based on the logic of your application. The following table lists the names of the existing APIs and respective superseding APIs.

Deprecated API	New API
ProSolidSurfaceVisit()	ProSolidBodiesCollect() ProSolidyBodySurfaceVisit()
ProPartToProIntfData()	ProPartToProInterfaceData() ProQuiltdataTypeGet() ProQuiltdataTypeSet()
ProXsecGeometryCollect()	ProXSectionItemsCollect() ProXSectionItemDataGet()
ProXsecPlanarWithoptionsCreate()	ProXSectionPlanarCreate()
ProXsecExcludeCompGet()	ProXSectionExcludeCompGet()
ProXsecCompXhatchStyleGet()	ProXsectionCompXhatchStyleGet() ProXSectionItemXhatchStyleGet()
ProXsecCompNewXhatchStyleGet()	ProXsectionCompXhatchStyleGet ()

Deprecated API	New API
	ProXSectionItemXhatchStyleGet()
ProXsecCompXhatchStyleSet()	ProXsectionCompXhatchStyleSet() ProXSectionItemXhatchStyleSet()
ProXsecCompNewXhatchStyleSet()	ProXsectionCompXhatchStyleSet() ProXSectionItemXhatchStyleSet()
ProXsecCompNewXhatchStyleSetByName()	ProXsecNewXhatchStyleCreateFromName() /ProXsectionCompXhatchStyleSet() ProXSectionItemXhatchStyleSet()
ProXsecOffsetCreate()	ProXSectionOffsetCreate()
ProPartDensitySet()	ProMaterialCurrentSet() ProMaterialPropertySet()

The following points are important to remember:

- Check instances of deprecated API use.
- Modify code to use superseding APIs
- Test the updated application on legacy models and on multibody models.
- Some of the deprecated APIs work on multibody models if the special configuration option `allow_gmb_tkapi` is set to `yes`.

User-Defined Features

With the introduction of bodies in Creo Parametric 7.0.0.0, the following occurs:

- When creating a part, you can add body references to some features, such as to protrusion and cut features.
- When creating an assembly, you cannot add body references.
- UDFs created in an earlier release of Creo Parametricdo not have body references.
- UDFs created in an assembly in Creo Parametric 7.0.0.0 do not have body references.

For more information about how UDF placement is handled in Creo Parametric 7.0.0.0, refer to the section Multibody support in UDF and Copy feature on page 162 in the Core: Features on page 131 chapter.

Update in API Implementation to Support Multibody

Implementation of some APIs is changed. To programmatically select bodies, pass the string `3d_body` to the API `ProSelect` to select bodies.

Update in Existing Element Trees

Some existing features are affected due to the introduction of the multibody features such as creating or adding a body. The following header files are updated:

- `ProShell.h`
- `ProRib.h`
- `ProSmtDrvSurf.h`
- `ProExtrude.h`
- `ProRevolve.h`
- `ProSweep.h`
- `ProHole.h`
- `ProRound.h`
- `ProChamfer.h`
- `ProModifyRound.h`
- `ProModifyChamfer.h`
- `ProSmtShell.h`

The following points are important to remember:

- Check if the affected features are created in the Creo Parametric TOOLKIT application.
- If you are working in the single body or legacy environment, then no action is required.
- If you are working in the multibody environment, update the application code and use the updated element trees for the specific features.
- If you are using an updated version of the Creo Parametric TOOLKIT application, perform subsequent testing for multibody models.

New Element Trees for Supporting Multibody Features

To support multibody features, the following header files are added:

- `ProSplitBody.h`
- `ProBooleanBodies.h`
- `ProRemoveBody.h`
- `ProBodyCopy.h`
- `ProBodyOpts.h`

You can create multibody features using these new feature element trees based on the logic of your Creo Parametric TOOLKIT application.

Update to Values of Enumerated Data Types

With the introduction of the multibody environment, the following enumerated values are impacted:

- `PRO_SURFCOLL_ALL_SOLID_SRFS` is deprecated

> **Note**
>
> As a result, the functions `ProSrfcollectionRegenerate()`, `ProElementCollectionSet()` , and `ProSelbufferCollectionAdd()` return the error `PRO_TK_MULTIBODY_UNSUPPORTED` for multibody based models.

- The new enum values `PRO_SURFCOLL_BODY_SRFS` and `PRO_SURFCOLL_ALL_BODY_SRFS` are added.
- New enum value`PRO_LAYER_BODY` is added.

If the enum value `PRO_SURFCOLL_ALL_SOLID_SRFS` exists in your Creo Parametric TOOLKIT application code, then you need to update the code and test the legacy and multibody models.

Impact on Existing Structures

`ProQuiltData` is updated with a new structure member to support multibody environment. As a result, the function `ProPartToIntfData()` is impacted.

E

Creo Parametric TOOLKIT Registry File

This appendix describes how to use the Registry file to have a foreign program communicate with Creo Parametric.

Registry File Fields

The following table lists the fields in the registry file `creotk.dat` or `protk.dat`.

Field	Description
`name`	Assigns a unique name to the Creo Parametric TOOLKIT application. The name is used to identify the application if there is more than one. The name can be the product name and does not have to be the same as the executable name. This field has a limit of `PRO_NAME_SIZE-1` wide characters (`wchar_t`).
`startup`	Specifies the method Creo Parametric should use to communicate with the Creo Parametric TOOLKIT application. This field can take one of three values; `spawn`, `dll` or `java`. • `spawn`— If the value is `spawn`, Creo Parametric starts the foreign program using interprocess communications. • `dll` – If the value is `dll`, Creo Parametric loads the foreign program as a DLL. • `java` – If the value is `java`, Creo Parametric starts the application as a J-Link class. Consult the *J-Link User's Guide* for more details. The default value is `spawn`.
`fail_tol`	Specifies the action of Creo Parametric if the call to `user_initialize()` in the foreign program returns non-zero, or if the foreign program subsequently fails. If this is `TRUE`, Creo Parametric continues as normal. If this field is missing or is set to `FALSE`, Creo Parametric shuts down Creo Parametric and other foreign programs.
`exec_file`	Specifies the full path and name of the file produced by compiling and linking the Creo Parametric TOOLKIT application. In `DLL` mode, this is a dynamically linkable library; in `spawn` mode, it is a complete executable. This field has a limit of `PRO_PATH_SIZE-1` wide characters (`wchar_t`).
`text_dir`	Specifies the full path name to text directory that contains the language-specific directories. The language-specific directories contain the message files, menu files, resource files and UI bitmaps in the language supported by the Creo Parametric TOOLKIT application. Please refer to the User Interface: Menus, Commands, and Popupmenus on page 301 and User Interface: Messages on page 284 chapters for more information. The `text_dir` does not need to include the trailing `/text`; it is added automatically by Creo Parametric. The search priority for messages and menu files is as follows: 1. Current working directory 2. `text_dir\text` 3. `<creo_loadpoint>\<datecode>\Common Files\$<machine type>\text`, where `<machine_type>` is the

Field	Description
	machine-specific subdirectory, such as, `i486_nt` or `x86e_win64`. Set the environment variable `PRO_MACHINE_TYPE` to define the type of machine on which Creo Parametric is installed. The `text_dir` should be different from the Creo Parametric text tree. This field has a limit of `PRO_PATH_SIZE-1` wide characters (`wchar_t`).
`rbn_path`	Specifies the name of the ribbon file along with its path, which must be loaded when you open Creo Parametric. The location of the ribbon file is relative to the location of the text directory. The field `text_dir` specifies the path for the text directory. For example, if you want to specify a ribbon file `app1_rbn.rbn` placed at `text_dir/app1/app1_rbn.rbn`, specify `rbn_path` as `app1/app1_rbn.rbn`. If the field is not specified, by default, the ribbon file with its location, `text_dir/toolkitribbonui.rbn` is used.
`delay_start`	If you set this to `TRUE`, Creo Parametric does not invoke the Creo Parametric TOOLKIT application as it starts up, but enables you to choose when to start the application. If this field is missing or is set to `FALSE`, the Creo Parametric TOOLKIT application starts automatically.
`description`	Acts as a help line for your auxiliary application. If you leave the cursor on an application in the Start/Stop GUI, Creo Parametric displays the description text (up to 80 characters). You can use non-ASCII characters, as in menu files. To make the description appear in multiple languages, you must use separate `protk.dat` files in `<hierarchy>/<platform>/<text>/<language>`.
`allow_stop`	If you set this to `TRUE`, you can stop the application during the session. If this field is missing or is set to `FALSE`, you cannot stop the application, regardless of how it was started.
`end`	Indicates the end of the description of the Creo Parametric TOOLKIT application. It is possible to add further statements that define other foreign applications. All of these applications are initialized by Creo Parametric.

Sample Registry Files

This section lists several examples that illustrate the various ways to have a foreign program communicate with Creo Parametric.

Example 1

In this example, Creo Parametric spawns the foreign program, which runs on the same machine. The communication is via pipes (the default mode when the foreign program runs on the same machine as Creo Parametric).

File: `protk.dat`

[Start of file on next line]

```
name        Product1
exec_file   /home/protk/$<machine_type>/obj/frnpgm1
```

```
text_dir    /home/protk
end
```

[End of file on previous line]

Example 2

This example illustrates how to run multiple foreign programs, as specified in the `protk.dat` file.

File: `protk.dat`

[Start of file on next line]

```
name         Product1
startup      dll
exec_file    /home/protk/$<machine_type>/obj/frnpgm1.dll
text_dir     /home/protk
end
name         Product2
startup      spawn
exec_file    /home/protk2/$<machine_type>/obj/frnpgm2
text_dir     /home/protk2
end
```

[End of file on previous line]

F

Creo Parametric TOOLKIT Library Types

This appendix describes the various libraries available in a Creo Parametric TOOLKIT installation.

Overview

The libraries available in a Creo Parametric TOOLKIT installation have been classified under:

- Standard Libraries on page 2105
- Alternate Libraries on page 2105

From Creo Parametric 4.0 F000 onward, the libraries listed in the following table are no longer supported and will not be available with the software. The New Library Name column provides a list of the equivalent libraries that are now available. Apart from the compatibility issues explained in the chapter Version Compatibility: Creo Parametric and Creo Parametric TOOLKIT on page 41, applications based on Creo Parametric 3.0 and previous releases will continue to run successfully with Creo Parametric 4.0.

Old Library Name	New Library Name
protk_dll.lib	protk_dll_NU.lib
protoolkit.lib	protoolkit_NU.lib
protk_dllmd.lib	protk_dllmd_NU.lib
protkmd.lib	protkmd_NU.lib

Linking the Applications

Before you run an existing application in Creo Parametric 8.0, link it to the new libraries and the import libraries: `ucore.lib` and `udata.lib`. The new libraries do not link the application to the Unicode libraries but use `ucore.lib` and `udata.lib` at runtime to resolve the Unicode dependencies, which also reduces the size of the application. Since these libraries are import libraries, the application must resolve Unicode dependencies at runtime by loading the actual libraries `ucore64.dll` and `udata64.dll`. These dlls are located at `<creo_load_point>/Common Files/<platform>/obj` and `<creo_load_point>/Common Files/<platform>/lib`.

For synchronous applications, the references to `ucore64.dll` and `udata64.dll` are resolved by Creo Parametric when the application is started.

For asynchronous applications, these references must be resolved by the application. For linking asynchronous applications, add the path where the dlls `ucore64.dll` and `udata64.dll` are located, that is `<creo_load_point>/Common Files/<platform>/lib` to the environment variable `PATH`.

All the sample makefiles available with Creo Parametric TOOLKIT use the new libraries. For instance, the sample example `make_examples` created for Creo Parametric TOOLKIT applications created for Creo Parametric TOOLKIT

applications contains information on how to use `protk_dll_NU.lib`. The sample file is located at `<creo_toolkit_loadpoint>/<platform>/obj`.

Standard Libraries

Most Creo Parametric TOOLKIT users will be able to use the standard Creo Parametric TOOLKIT libraries. These libraries are available on all platforms and are used by the majority of Creo Parametric TOOLKIT sample applications.

Library Name	Purpose
`protoolkit_NU.lib ucore.lib udata.lib`	Spawn mode library
`pt_asynchronous.lib`	Asynchronous mode library
`protk_dll_NU.lib ucore.lib udata.lib`	DLL mode library

Alternate Libraries

Creo Parametric TOOLKIT offers alternate libraries that may be useful for applications compiled with `/MD` flag and built with `msvcrt.lib`. These libraries are similar to the standard Creo Parametric TOOLKIT libraries in content, but differ in using `msvcrt.lib` instead of `libcmt.lib`.

Library Name	Purpose
`protkmd_NU.lib`	Spawn mode library
`ptasyncmd.lib`	Asynchronous mode library
`protk_dllmd_NU.lib`	DLL mode library

The makefiles `make_install_md` and `make_async_md` build with these libraries.

Note

Although `/MD` provides compatibility with multi-threaded components, Creo Parametric TOOLKIT calls must be made within a single thread. Creo Parametric does not respond to calls made from multiple threads. Extra threads may be created by applications only to do tasks which do not directly call Creo Parametric TOOLKIT functions.

G

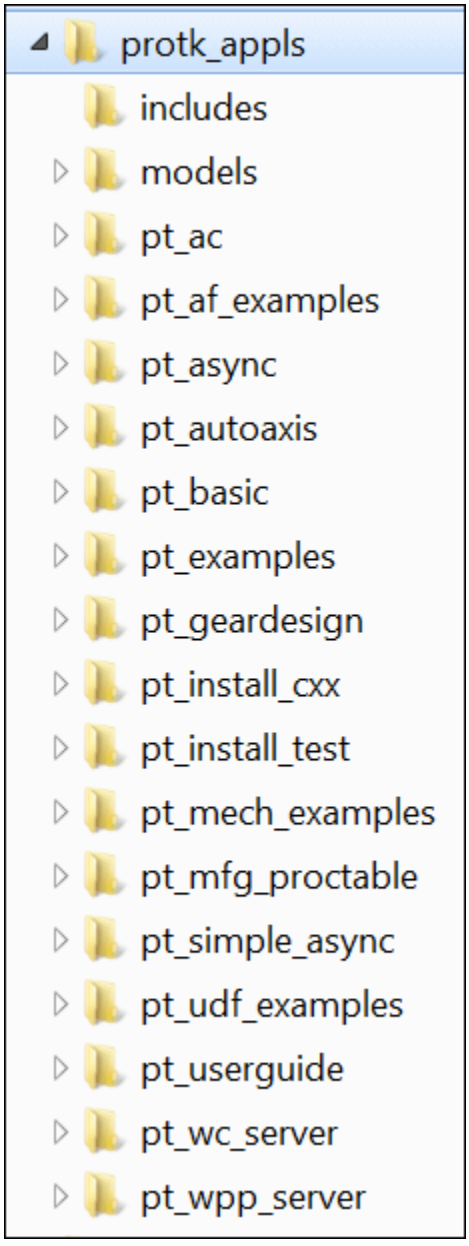
Creo Parametric TOOLKIT Sample Applications

This appendix describes the sample applications provided with Creo Parametric TOOLKIT.

Installing Sample Applications

When you install Creo Parametric TOOLKIT from the Creo Parametric CD, Creo Parametric TOOLKIT is installed under the loadpoint of Creo Parametric, that is, `<creo_loadpoint>\<datecode>\Common Files\protoolkit\protk_appls`. In Creo Parametric 6.0.0.0 and later, these sample applications are digitally signed. Refer to the Fundamentals on page 22 chapter for more information on Installation of Creo Parametric TOOLKIT.

The Creo Parametric TOOLKIT directory contains all the headers, libraries, example applications, and documentation specific to Creo Parametric TOOLKIT. The following diagram illustrates the applications installed under the `protk_appls` directory after installation.



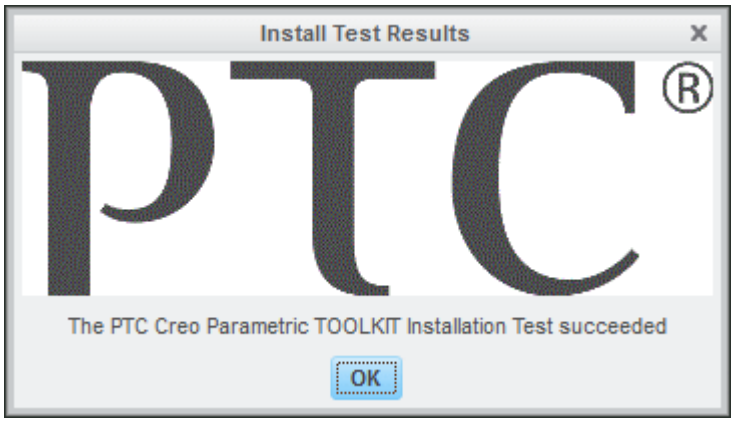
Details on Sample Applications

The sample applications provided with Creo Parametric TOOLKIT are available in directories under the following path `<creo_loadpoint>\<datecode>\Common Files\protoolkit\protk_appls`.

pt_inst_test

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_install_test	make_install

The application pt_inst_test is used to check the Creo Parametric TOOLKIT Installation. It verifies the ProMenubar and custom user interface dialog box functions.



pt_inst_cxx

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_install_cxx	make_install_cxx

The application pt_inst_cxx is used to check the C++ version of the Creo Parametric TOOLKIT installation. It verifies the ProMenubar and custom user interface dialog box functions that use the C++ compiler and classes.

pt_inst_test_md

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_install_test	make_install_md

The application pt_inst_test_md provides the version of the makefile for Windows platforms that uses MD libraries.

MD libraries are intended for building a DLL for Windows. Some Microsoft libraries must be linked with these libraries. For more information on MD libraries, refer to Alternate Libraries on page 2105.

pt_autoaxis

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_autoaxis	make_autoaxis

The sample application pt_autoaxis automatically creates axes on revolved surfaces if they don't already exist. It is intended for manufacturing engineers who receive a model with holes which were not made with standard Creo Parametric hole features. It covers functions for feature creation and geometry analysis.

pt_userguide

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_userguide	make_userguide

The sample application pt_userguide consolidates examples that access the User Interface.

pt_examples

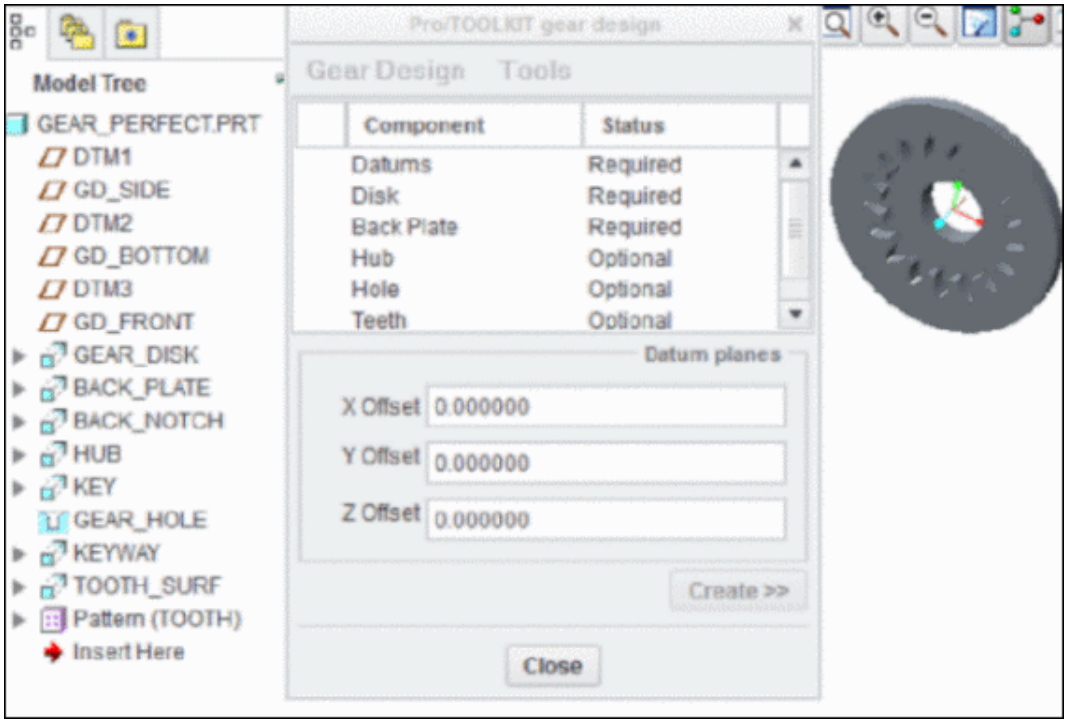
Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_examples	make_examples

The sample application pt_examples consists of Creo Parametric TOOLKIT application examples. It provides a user interface to access many areas of Creo Parametric TOOLKIT. This application covers Creo Parametric TOOLKIT functions and modules, including ProMenubar and ProMenu based UI functions. The directory pt_examples includes sub-directories containing useful utility functions.

pt_geardesign

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_geardesign	make_geardesign

The sample application pt_geardesign provides the user interface to create gear models. This application covers feature creation using element trees and the custom user interface dialog box functions.



pt_async

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_async	make_async

The sample application `pt_async` provides an example for the full asynchronous mode.

pt_async_md

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_async	make_async_md

The sample application `pt_async_md` provides an example for the asynchronous mode compilation of a DLL using MD libraries on Windows. The application requires two binaries, namely, `pt_async_md.dll`, and the wrapper executable `pt_async_md_wrapper.exe` which will load the DLL and invoke it. The application is identical to `pt_async`in all other respects. For more information on MD libraries, refer to Alternate Libraries on page 2105.

pt_simple_async

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_simple_async	make_simple_async

The sample example pt_simple_async provides an example for the simple asynchronous mode.

pt_basic

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_basic	make_basic

The sample application pt_basic provides the Creo Parametric TOOLKIT application template. It verifies the user interface, notifications, and the application setup.

pt_af_examples

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_af_examples	make_af_examples

The sample application pt_af_examples provides production examples using annotation features and annotations.

pt_udf_examples

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_udf_examples	make_udf_examples

The sample application pt_udf_examples demonstrates production examples using UDF placement capabilities. These include a flexible user interface for placing UDF libraries interactively, by reading the properties of the UDF library file.

pt_mech_examples

Location	Makefile
<creo_toolkit_loadpoint>/protk_appls/pt_mech_examples	make_mech_examples

The sample application pt_mech_examples provides production examples using the ability to assign, create and modify Creo Simulate item properties.

H

Advanced Licensing Options

This chapter describes the licensing requirements for advanced options in Creo Parametric TOOLKIT.

Advance Licensing Options for Creo Parametric TOOLKIT

To use some of the functionality in Pro/TOOLKIT you must have advanced development license options.

For each TOOLKIT function that requires an advanced license, theCreo Parametric TOOLKIT header file entry includes a note specifying the development license requirement. This note is also visible in the APIWizard description page for the function. Advanced licenses are required in the following situations:

- To run a locked application, Creo Parametric requires the basic Creo Parametric TOOLKIT development option and any advanced toolkit options required by specific functions called by the application. If the application contains calls to such functions, Creo Parametric checks out the corresponding advanced license option on demand.
- To unlock an application, the unlock utility requires the basic Creo Parametric TOOLKIT development option and any advanced toolkit options required by specific functions called by the application. The utility will not hold any of the advanced options, as it does the basic Creo Parametric TOOLKIT development option, after unlock is completed.
- Creo Parametric does not require any of the Creo Parametric TOOLKIT licenses to run a properly unlocked application.

Applications are assigned requirements for advanced options based on whether the application is coded to use any functions requiring the advanced option. It does not matter if an application does not use the function requiring licensing during a particular invocation of the application. The licensing requirements are resolved the moment the application is started by or connects to Creo Parametric, not at the first time an advanced function is invoked.

For more information on how to unlock an application, refer to the section Unlocking a Creo Parametric TOOLKIT Application on page 44.

Pro/DEVELOP to Creo Parametric TOOLKIT Function Mapping

This appendix describes how to update legacy applications using Pro/DEVELOP functions with current Creo Parametric TOOLKIT functions.

From Creo Parametric 2.0 onward, the Pro/Develop functions are obsolete. The Pro/Develop header files and related support files will not be shipped with Creo Parametric in future. PTC recommends that you update applications that use Pro/Develop functions to use equivalent Creo Parametric TOOLKIT functions or Creo Object TOOLKIT C++ methods.

The Relationship Between Creo Parametric TOOLKIT and Pro/DEVELOP

Creo Parametric TOOLKIT replaces and contains Pro/DEVELOP, the customization toolkit until Release 17 of Pro/ENGINEER. Creo Parametric TOOLKIT uses an Object-Oriented style.

Creo Parametric TOOLKIT OHandles:

Creo Parametric TOOLKIT OHandles are equivalent to the type `Prohandle` used in Pro/DEVELOP. You can convert the handles between Creo Parametric TOOLKIT and Pro/DEVELOP simply by casting to the appropriate type. Creo Parametric TOOLKIT provides different OHandles for different object types where Pro/DEVELOP provided only a single generic handle; this provides for better type-checking during compilation. See Converting from Pro/DEVELOP on page 2116 for more details.

Converting from Pro/DEVELOP

You can convert functions from Pro/DEVELOP to Creo Parametric TOOLKIT, and also mix the two styles of functions.

Using Pro/DEVELOP Applications with Creo Parametric TOOLKIT

Creo Parametric TOOLKIT replaces Pro/DEVELOP and provides most of the functionality that existed in Pro/DEVELOP. Existing Pro/DEVELOP applications will not become obsolete however, for the following reasons:

- Creo Parametric TOOLKIT inherits from Pro/DEVELOP the mechanisms by which the application C code is integrated into Creo Parametric. These mechanisms will continue to be used by Creo Parametric TOOLKIT for the indefinite future.
- The complete library of Pro/DEVELOP functions is installed automatically along with the library of Creo Parametric TOOLKIT functions, and will be installed in this way from Release 2000i onwards.

Therefore, Pro/DEVELOP applications built using Release 17 will continue to work with Pro/ENGINEER Release 18 and later, without having to be recompiled and relinked. Using Creo Parametric TOOLKIT, you can recompile and relink Pro/DEVELOP applications developed using Release 17 without having to change the source code. These applications will continue to function as before.

However, you should plan to convert your applications to Creo Parametric TOOLKIT as soon as possible, even if you do not need to use any of the new functionality provided by Creo Parametric TOOLKIT. The conversion is desirable

because Creo Parametric TOOLKIT provides more consistent and complete functionality, even in areas already well-covered by Pro/DEVELOP. In addition, PTC will give lower priority to requests for enhancements and maintenance to Pro/DEVELOP functions than to requests for equivalent Creo Parametric TOOLKIT functions, where they exist.

Thanks to the technology they share, you can use functions from both Creo Parametric TOOLKIT and Pro/DEVELOP within a single application. This means that:

- You can convert a Pro/DEVELOP application to use Creo Parametric TOOLKIT functions gradually.
- Pro/DEVELOP applications can use the new functionality provided by Creo Parametric TOOLKIT without the immediate need for a complete conversion.

A final reason for wanting to mix Pro/DEVELOP and Creo Parametric TOOLKIT functions is that not all of the Pro/DEVELOP functions have been replaced by equivalent Creo Parametric TOOLKIT functions.

Techniques of Conversion and Mixing

Besides a difference in the conventions they use, Pro/DEVELOP and Creo Parametric TOOLKIT reference items in the Creo Parametric database in different ways. The following sections describe the technical points to consider when you convert from Pro/DEVELOP to Creo Parametric TOOLKIT, or when you mix both types of functions in a single application.

Terminology

In general, the terminology used by Creo Parametric TOOLKIT is close to that of Pro/DEVELOP. The following table lists the most important terms that differ in meaning between the two toolkits.

Creo Parametric TOOLKIT	Pro/DEVELOP
Object	N/A
Model	Object
Solid (part or assembly)	Model
Surface	Face or surface
Component of an assembly	Member
Component path	Member identifier table (`memb_id_tab`)
External data	Generic application data

General Functionality

To find the functions in Creo Parametric TOOLKIT that cover a particular area of functionality, scan the appendix in this user's guide, or use the Topical option in the Creo Parametric TOOLKIT browser. Beware of any difference in terminology from Pro/DEVELOP identified in the previous section.

If you want to find the Creo Parametric TOOLKIT equivalent of a particular Pro/DEVELOP function, refer to the table Equivalent Pro/DEVELOP Functions on page 2128. The table maps each Pro/DEVELOP function to the closest equivalent Creo Parametric TOOLKIT function (or functions).

In some functional areas, especially where Pro/DEVELOP provided good coverage, you can use the equivalent Creo Parametric TOOLKIT functions in an identical way, although the function names, return values, and sometimes the order of the arguments have been changed to conform to Creo Parametric TOOLKIT conventions.

For example, the following Pro/DEVELOP functions are almost exactly equivalent to the Creo Parametric TOOLKIT functions listed.

Pro/DEVELOP Function	Creo Parametric TOOLKIT Function
`promenu_create()`	`ProMenuFileRegister()`
`promenu_expand()`	`ProMenuAuxfileRegister()`
`promenu_on_button()`	`ProMenubuttonActionSet()`

Other functions require more care, however. For example, one of the conventions of Creo Parametric TOOLKIT is that the input arguments come before the output arguments.

In some areas of functionality, traditional Pro/DEVELOP techniques have been replaced in Creo Parametric TOOLKIT by techniques that are more general, flexible, and consistent with the techniques used within Creo Parametric. A good example is the visit functions, which replace two different Pro/DEVELOP techniques. For example:

Pro/DEVELOP Function	Creo Parametric TOOLKIT Equivalent
`prodb_get_feature_ids()`	`ProSolidFeatVisit()`
`prodb_first_part_face()`, `prodb_next_part_face()`	`ProSolidBodySurfaceVisit()`

It is possible to use the Creo Parametric TOOLKIT visit functions to create a utility that follows one of the Pro/DEVELOP styles. An example is shown in the section Expandable Arrays on page 59.

Some areas of Creo Parametric TOOLKIT functionality reveal a more general, and more consistent, view of the contents of the Creo Parametric database than that familiar to users of Pro/DEVELOP, and therefore require a slightly deeper understanding. For example, Creo Parametric TOOLKIT does not contain exact equivalents of the following Pro/DEVELOP functions for traversing the components of an assembly:

- `prodb_first_member()`
- `prodb_next_member()`

Assembly components (called “members” in Pro/DEVELOP) are represented as features in the Creo Parametric database, so these two functions can be replaced by a call to `ProSolidFeatVisit()`, using `ProFeatureTypeGet()` to identify the features of type `PRO_FEAT_COMPONENT`. The feature identifier for an assembly component is identical to the member identifier used in Pro/DEVELOP.

In the same way, the following Pro/DEVELOP functions that find datum planes and datum curves are also replaced by more generic functions in Creo Parametric TOOLKIT:

- `prodb_first_datum()`
- `prodb_next_datum()`
- `prodb_get_datum_curves()`

Here, too, the first step is to traverse the features using `ProSolidFeatVisit()`. You can then traverse all the geometrical items in a feature using `ProFeatureGeomitemVisit()`. Datum planes are geometry items of type `PRO_SURFACE`, in features of type `PRO_FEAT_DATUM`; datum curves are geometry items of type `PRO_CURVE`, which can occur in features of many types.

This manual always explains the structure of the Creo Parametric database wherever necessary, without assuming any prior knowledge of the Pro/DEVELOP viewpoint. As shown in the previous examples, if you are converting a Pro/DEVELOP application that traverses Creo Parametric geometry, you should pay particular attention to the Core: 3D Geometry on page 170 appendix.

You can use Creo Parametric TOOLKIT functions to create utilities for the specific cases you need. Many such utilities are provided in the sample code located under the Creo Parametric TOOLKIT loadpoint.

Finally, Creo Parametric TOOLKIT covers whole new areas of functionality that were not supported at all by Pro/DEVELOP, such as the direct programmatic creation of features, including simple kinds of sketched features, datum planes, and manufacturing features. Some Pro/DEVELOP applications, especially those that create features using user-defined features (UDFs), and which customize Manufacturing, may therefore benefit from a complete redesign to take full advantage of Creo Parametric TOOLKIT.

Registry Files

The Creo Parametric TOOLKIT registry file has the same format as the Pro/DEVELOP registry file. The search path used by Creo Parametric TOOLKIT to find the registry file is like that used by Pro/DEVELOP. However, the file name `prodev.dat` is now replaced by `creotk.dat` or `protk.dat`, and the

configuration file option `prodevdat` is now either `creotkdat`, or `protkdat`, or `toolkit_registry_file`. To convert from Pro/DEVELOP to Creo Parametric TOOLKIT, simply substitute these names.

For an extended period, the search path for the Pro/DEVELOP registry file will continue to be used by Creo Parametric, in addition to the search path for Creo Parametric TOOLKIT. Therefore, you do not need to rename the Pro/DEVELOP registry file or configuration file option immediately.

Menu and Message Files

Although the Pro/DEVELOP functions for accessing menus and messages have been replaced by close equivalents in Creo Parametric TOOLKIT, the menu and message files themselves retain exactly the same form and function. No conversion is necessary.

Unlocking Your Application

The Creo Parametric TOOLKIT script for unlocking a finished application is named `protk_unlock`, but is otherwise identical to `prodev_unlock`.

Application Program Structure

All the Pro/DEVELOP run modes are available in identical form in Creo Parametric TOOLKIT, and the structure of a Creo Parametric TOOLKIT application is the same as that of a Pro/DEVELOP application. Some of the core functions have been given new Creo Parametric TOOLKIT-style names for the sake of consistency, but are otherwise the same. The functions `user_initialize()` and `user_terminate()` remain identical in name and purpose.

Handles and Data Types

Although Creo Parametric TOOLKIT is more rigorous than Pro/DEVELOP in the way it references objects in the Creo Parametric database, there are some close correspondences that simplify the task of mixing Pro/DEVELOP and Creo Parametric TOOLKIT functions.

As a general rule, database items referred to in Pro/DEVELOP by the type `Prohandle`, and referred to as OHandles (opaque handles) in Creo Parametric TOOLKIT, are pointers to the same Creo Parametric data structures. You can directly convert them by casting. The following table lists the most important examples.

Pro/DEVELOP Prohandle for the item type	Can be cast directly to the Creo Parametric TOOLKIT object
Object	`ProMdl`
Assembly	`ProAssembly`

Pro/DEVELOP Prohandle for the item type	Can be cast directly to the Creo Parametric TOOLKIT object
Part	`ProPart`
Model (part or assembly)	`ProSolid`
Surface	`ProSurface`
Contour	`ProContour`
Edge	`ProEdge`
Curve	`ProCurve`
Datum quilt	`ProQuilt`
Point	`ProPoint`
Axis	`ProAxis`
Coordinate system	`ProCsys`

For database items that can be identified in Pro/DEVELOP by an integer identifier, that identifier is the same one generated by Creo Parametric TOOLKIT functions such as `ProSurfaceIdGet()` and `ProEdgeIdGet()`, and is the same one required as input to functions such as `ProSurfaceInit()`. It is also the value of the `id` field when one of these objects is represented as a `ProGeomitem`.

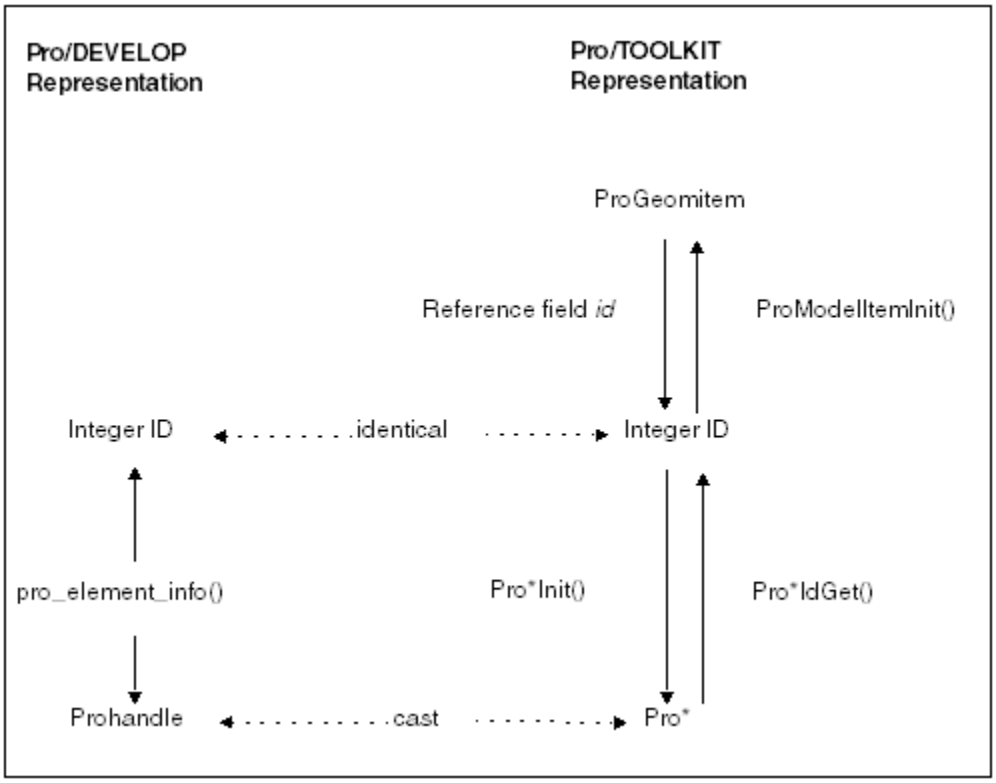
Converting a Pro/DEVELOP integer identifier to a Creo Parametric TOOLKIT OHandle can be done in two ways:

- Convert to a `Prohandle` within Pro/DEVELOP using `pro_element_info()`, then cast the resulting pointer.
- Use the identifier directly as the input to the appropriate `Pro*Init()` function.

The following diagrams show the possible conversion paths between Pro/DEVELOP and Creo Parametric TOOLKIT for database items.

The first diagram applies to objects of type Surface, Edge, Axis, Csys, Curve, Point, and Quilt. In each case, replace the asterisk (*) with the appropriate name.

Pro/DEVELOP Database Item Conversion Path



The exception to the previous diagram is that the Pro/DEVELOP function `pro_element_info()` is not supported for coordinate system datums.

A contour does not have an integer identifier in either Pro/DEVELOP or Creo Parametric TOOLKIT, but you can convert the Pro/DEVELOP `Prohandle` to `ProContour` and back by casting.

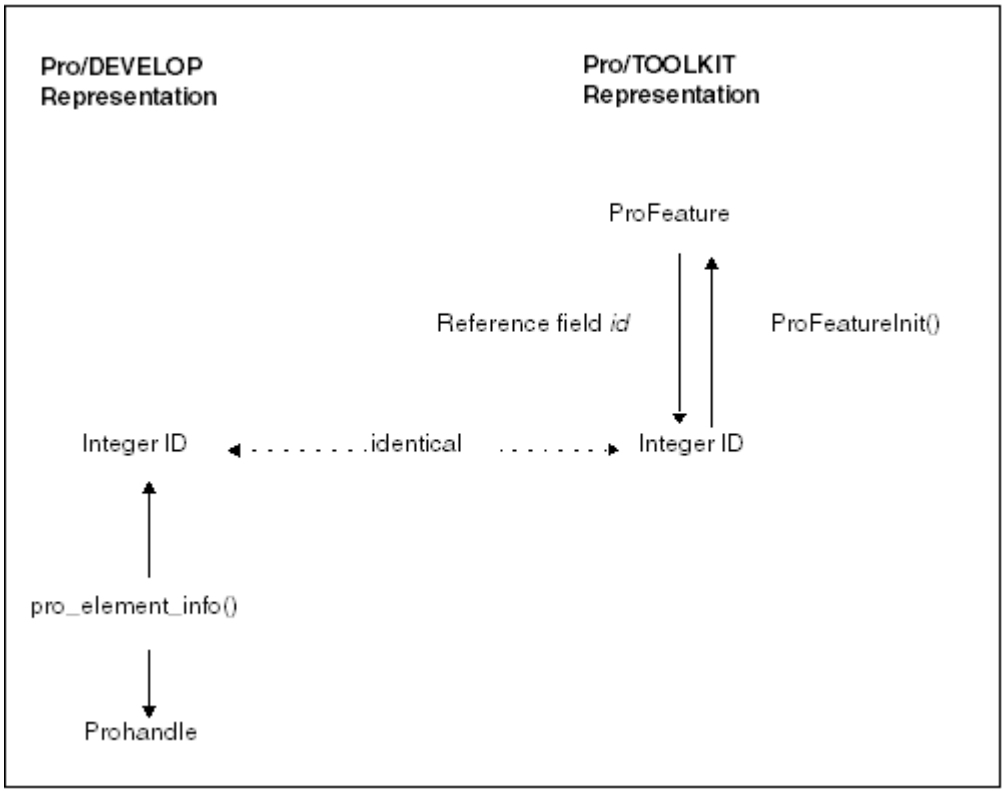
A view does not have an integer identifier in Creo Parametric TOOLKIT, but you can convert the Pro/DEVELOP `Prohandle` to `ProView` by casting.

A vertex is represented in Pro/DEVELOP as an edge (or a curve) and a value for the parameter `t`of either 0 or 1. Creo Parametric TOOLKIT uses this technique in the `ProSelection` object, but for function inputs and in `ProGeomitem` it uses the specific types `PRO_EDGE_START` and `PRO_EDGE_END` (and `PRO_CRV_START` and `PRO_CRV_END` for datum curve ends). Because `PRO_EDGE_START` and `PRO_CRV_START` always refer to the end where`t` = 0, conversion is easy.

A feature is represented in Creo Parametric TOOLKIT by `ProFeature`, which is a DHandle, and therefore not equivalent to a Pro/DEVELOP `Prohandle`. The integer identifier still maps directly, however.

The following diagram applies to converting features.

Feature Conversion



The following objects are DHandles, which are identical in form to `ProModelitem` and were identified only by an integer identifier in Pro/DEVELOP. They also inherit from ProModelitem, which means that, for example, `ProSelectionModelitemGet()` can be used to unpack them from a `ProSelection` object after calling `ProSelect()`. In each case, the id field in the object handle corresponds to the integer id used to identify these objects in Pro/DEVELOP.

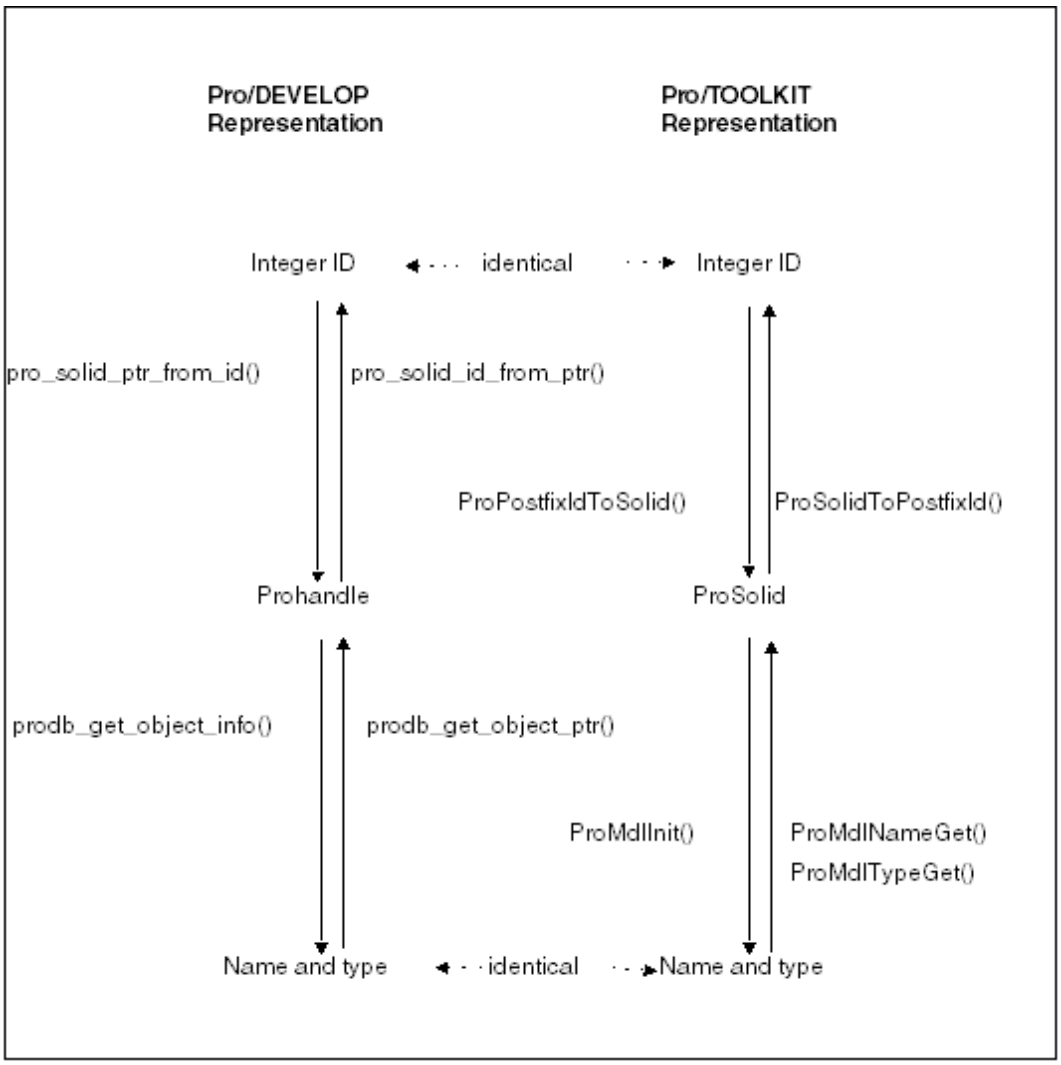
- `ProDimension`
- `ProGtol`
- `ProDtlnote`
- `ProDtlentity`
- `ProDtlsyminst`
- `ProDtlsymdef`
- `ProDtlgroup`
- `ProDgmitem`
- `ProNote`

Note

The objects `ProDtlnote` and `ProNote` refer to detail (drawing) notes, and notes respectively. They share the same value of the type field - `PRO_NOTE` - but they are distinguished by the type of their owning model.

The following diagram applies to Creo Parametric TOOLKIT objects `ProSolid`, `ProPart`, and `ProAssembly` when you map to Pro/DEVELOP parts and assemblies. For objects of type `ProMdl` that are not parts or assemblies, the integer identifiers are not applicable, but the rest of the diagram is correct.

Pro/DEVELOP Part and Assembly Mapping



Many explicit data types from Pro/DEVELOP have been carried across into Creo Parametric TOOLKIT directly, and, although they generally have been given new names, they are directly compatible. In fact, the remaining Pro/DEVELOP include files now reference the new definitions in the Creo Parametric TOOLKIT include files.

Enumerated types have also been given new names for their values, and in some cases where Pro/DEVELOP used `#defined` values of an integer, Creo Parametric TOOLKIT provides an enum. However, the integer mapping of the values is retained in every case. Some examples of Pro/DEVELOP data types now defined in Creo Parametric TOOLKIT are shown in the following table.

Pro/DEVELOP	Creo Parametric TOOLKIT
`Ptc_surf`	`ProSurfacedata`
`Ptc_curve`	`ProCurvedata`
`Pro_linestyle`	`ProLinestyle`
`pro_mode`	`ProMode`
`Pro_text_attributes`	`ProTextAttribute`
`int`	`ProMousebutton`
`int`	`ProDrawMode`
`int`	`ProColortype`

For some items that have representations in both Creo Parametric TOOLKIT and Pro/DEVELOP, the data structures are not the same, and no direct conversion is possible. However, you can always convert by reducing such structures to their component data items. For example, to convert the Pro/DEVELOP representation of a parameter, `Pro_parameter_info`, to the Creo Parametric TOOLKIT `ProParameter` object, you can use functions such as `ProParameterInit()` and `ProParameterValueWithUnitsSet()`, using the fields in the Pro/DEVELOP structure as inputs. (In this case, it would probably be better to make a complete conversion to the Creo Parametric TOOLKIT functions, and thus avoid mixing these types in an application.)

A more complex example of this is the Pro/DEVELOP `Select3d` structure, whose Creo Parametric TOOLKIT counterpart is `ProSelection`. The following table explains the mapping by showing the fields of `Select3d` alongside the Creo Parametric TOOLKIT functions that extract the equivalent information from `ProSelection`.

Select3d Field	Creo Parametric TOOLKIT Read Access
`sel_type`	`ProSelectionModelitemGet()`, then read `modelitem.type`. See the note that follows this table.
`selected_id`	`ProSelectionModelitemGet()`, then read `modelitem.id`.
`selected_ptr`	`ProSelectionModelitemGet()`, then `Pro*Init()`, depending on the type.
`select_pnt`	`ProSelectionPoint3dGet()`.
`sel_param`	`ProSelectionUvParamGet()`.
`sel_depth`	`ProSelectionDepthGet()`.

Select3d Field	Creo Parametric TOOLKIT Read Access
part_ptr	ProSelectionAsmcomppathGet(), and ProAsmcomppathMdlGet().
assembly_ptr	ProSelectionAsmcomppathGet(), then read comppath.owner.
memb_num	ProSelectionAsmcomppathGet(), then read comppath.table_num.
memb_id_tab	ProSelectionAsmcomppathGet(), then read comppath.comp_id_tab.
view_ptr	ProSelectionViewGet().

Note

The values of the Select3d field sel_type do not map directly to values of ProType, used in ProModelitem. Do not convert these types by direct assignment. See the next table for the mapping.

This table also makes clear what data from Pro/DEVELOP you need to build a ProSelection object in Creo Parametric TOOLKIT, using the functions ProSelectionAlloc(), ProSelectionSet(), ProSelectionUvParamSet(), and ProSelectionVerify().

There was an anomaly in selecting datum points in Pro/DEVELOP that has been corrected in Creo Parametric TOOLKIT. The selected_id is the identifier of the feature, and the datum point identifier is given by the field sel_elem_id. However, this does not carry over into Creo Parametric TOOLKIT—the ProModelitem identifier is the identifier of the datum point. (To get the feature, use the function ProGeomitemFeatureGet().)

The following table shows how the values of the sel_type field in Select3d (and the corresponding pro_select() option strings) map to values of ProType used in Creo Parametric TOOLKIT for ProModelitem and ProGeomitem objects used in Creo Parametric TOOLKIT as object handles that inherit from ProModelitem, such as ProGeomitem, ProDimension, ProGtol, ProDtlnote, and so forth.

pro_select() option	Select3d sel_type	ProType
point	SEL_3D_PNT	PRO_POINT
axis	SEL_3D_AXIS	PRO_AXIS
datum	SEL_3D_SRF	PRO_SURFACE
csys	SEL_3D_CSYS	PRO_CSYS
feature	SEL_3D_FEAT	PRO_FEATURE
edge	SEL_3D_EDG	PRO_EDGE
edge_end	SEL_3D_VERT	PRO_EDGE_START or PRO_EDGE_END
curve	SEL_3D_CURVE	PRO_CURVE

pro_select() option	Select3d sel_type	ProType
curve_end	SEL_CURVE_END	PRO_CRV_START or PRO_CRV_END
sldedge	SEL_3D_EDGE	PRO_EDGE
qltedge	SEL_3D_EDGE	PRO_EDGE
surface	SEL_3D_SRF	PRO_SURFACE
sldface	SEL_3D_SRF	PRO_SURFACE
qltface	SEL_3D_SRF	PRO_SURFACE
dtmqlt	SEL_3D_SRF_LIST	PRO_QUILT
part	SEL_3D_PART	PRO_PART
prt_or_asm	SEL_3D_PART	PRO_PART or PRO_ASSEMBLY
dimension	DTL_DIM	PRO_DIMENSION
ref_dim	DTL_REFDIM	PRO_REF_DIMENSION
gtol	DTL_GTOL	PRO_GTOL
dtl_symbol	DTL_SYMBOL	PRO_SYMBOL_INSTANCE
dwg_table	SEL_DWG_TABLE	PRO_DRAW_TABL
any_note	DTL_USER_NOTE	PRO_NOTE
note_3d	SEL_3D_NOTE	PRO_NOTE
dgm_obj	SEL_DGM_REF_OBJ	PRO_DIAGRAM_OBJECT
dgm_non_cable_wire	DTL_WIRE	PRO_DIAGRAM_WIRE
draft_ent	DTL_DRAFT_ENT	PRO_DRAFT_ENTITY
ext_obj	SEL_3D_EXT_OBJ	PRO_EXTOBJ
table_cell	SEL_TABLE_CELL	PRO_DRAW_TABLE_CELL

To minimize the need to convert between `Select3d` and `ProSelection`, follow these guidelines:

- If you need a `Select3d` as an input to a Pro/DEVELOP function, and the element referred to is to be selected interactively, use one of the Pro/DEVELOP select functions listed below instead of `ProSelect()` until you can convert the whole application.
  - `pro_select()`
  - `pro_get_selection()`
  - `pro_set_and_get_selection()`
- If you need to use a Pro/DEVELOP function whose output is, or contains, a `Select3d`, try to process the output using Pro/DEVELOP functions instead of converting to a `ProSelection` object where possible. For example, use `pro_show_select()` instead of `ProSelectionHighlight()`.

To help you maintain such mixtures, the online browser retains the description of the Pro/DEVELOP selection functions and `Select3d` that are, strictly speaking, superseded by Creo Parametric TOOLKIT.

However, PTC recommends that you retain your documentation for Pro/DEVELOP to use with Creo Parametric TOOLKIT.

Equivalent Pro/DEVELOP Functions

The following table lists the functions that have equivalents in Creo Parametric TOOLKIT. If the Pro/DEVELOP function is not included in this list, the function retains the Pro/DEVELOP style in Creo Parametric TOOLKIT. For ease of use, the Pro/DEVELOP functions are presented by functional group.

Pro/DEVELOP Function	Equivalent Creo Parametric TOOLKIT Function
Core Functions	
Synchronous Mode	
`pro_term()`	`ProEngineerEnd()`
Asynchronous Mode	
`prodev_start_proengineer()`	`ProEngineerStart()`
`prodev_set_interrupt_func()`	Not supported in Creo Parametric TOOLKIT.It can be safely removed from applications which call it. For information on how to structure a full asynchronous mode application to accept Creo Parametric events, refer to the chapter Core: Asynchronous Mode on page 277.
`prodev_handle_interrupt()`	`ProEventProcess()`
`prodev_set_proe_term_func()`	`ProTermFuncSet()`
`user_proe_term_func()`	`ProTerminationAction()`
`prodev_get_proe_status()`	`ProEngineerStatusGet()`
User-Supplied Main	
`prodev_main()`	`ProToolkitMain()`
Menus	
Adding a Menu Button	
`promenu_create()`	`ProMenuFileRegister()`
`promenu_expand()`	`ProMenuAuxfileRegister()`
`promenu_on_button()`	`ProMenubuttonActionSet()`
`promenu_load_action()`	`ProMenubuttonGenactionSet()`
New Menus	
`promenu_action()`	`ProMenuProcess()`
`promenu_exit_up()`	`ProMenuDelete()`
`promenu_make()`	`ProMenuCreate()`
`promenu_no_exit()`	`ProMenuHold()`
`promenu_exit_action_up()`	`ProMenuDeleteWithStatus()`
`promenu_make_compound()`	`ProCompoundmenuCreate()`
Preempting Creo Parametric Commands	
`promenu_load_pre_func()`	`ProMenubuttonPreactionSet()`
`promenu_load_post_func()`	`ProMenubuttonPostactionSet()`
Manipulating Menus	
`promenu_set_item_location()`	`ProMenubuttonLocationSet()`
`promenu_set_item_visible()`	`PropMenubuttonVisibilitySet()`
`promenu_remove_item()`	`ProMenubuttonDelete()`
Data Menus	

promenu_set_mode()	ProMenuModeSet()
promenu_set_data_mode()	ProMenuDatamodeSet()
Setting Menu Buttons	
promenu_set_item()	ProMenubuttonHighlight()
promenu_reset_item()	ProMenubuttonUnhighlight()
Controlling Accessibility of Menu Buttons	
promenu_make_item_ accessible()	ProMenubuttonActivate()
promenu_make_item_ inaccessible()	ProMenubuttonDeactivate()
Pushing and Popping Menus	
promenu_is_up()	ProMenuVisibilityGet()
promenu_push()	ProMenuPush()
promenu_pop()	ProMenuPop()
Run-Time Menus	
pro_select_strings()	ProMenuStringsSelect()
Entering Creo Parametric Commands	
proload_cmd_sequence()	ProMacroLoad()
promenu_push_command()	ProMenuCommandPush()
Message Window	
Writing a Message	
promsg_print()	ProMessageDisplay()
promsg_clear()	ProMessageClear()
Writing a Message to an Internal Buffer	
promsg_sprint()	ProMessageToBuffer()
Getting Keyboard Input	
promsg_getint()	ProMessageIntegerRead()
promsg_getdouble()	ProMessageDoubleRead()
promsg_getstring()	ProMessageStringRead()
promsg_getpasswd()	ProMessagePasswordRead()
Graphics and Object Display	
Manipulating Windows	
pro_clear_window()	ProWindowClear()
pro_view_repaint()	ProWindowRepaint()
pro_refresh_window()	ProWindowRefresh()
pro_get_current_window()	ProWindowCurrentGet()
pro_set_current_window()	ProWindowCurrentSet()
pro_open_object_window()	ProObjectwindowMdlnameCreate()
pro_close_object_window()	ProWindowDelete()
Model Orientation	
pro_get_cur_window_matrix()	ProWindowCurrentMatrixGet()
pro_get_view_matrix()	ProViewMatrixGet()
pro_set_view_matrix()	ProViewMatrixSet()
pro_reset_view()	ProViewReset()

pro_rotate_view()	ProViewRotate()
pro_store_view()	ProViewStore()
pro_retrieve_view()	ProViewRetrieve()
pro_get_view_names()	ProViewNamesGet()
Displaying Creo Parametric Objects	
progr_display_object()	ProSolidDisplay() (for parts and assemblies) ProMdlDisplay()
pro_show_select()	ProSelectionHighlight() ProSelectionDisplay() ProSelectionUnhighlight()
Graphics Colors and Line Styles	
progr_text_color()	ProTextColorSet()
progr_color()	ProGraphicsColorSet()
progr_get_color_map()	ProColormapGet()
progr_set_color_map()	ProColormapSet()
progr_set_line_style()	ProLinestyleSet()
progr_get_line_style_def()	ProLinestyleDataGet()
Displaying Graphics	
progr_move_3d()	ProGraphicsPenPosition()
progr_draw_3d()	ProGraphicsLineDraw()
progr_put_polyline()	ProGraphicsPolylineDraw()
progr_put_multi_polylines()	ProGraphicsMultiPolylinesDraw()
progr_put_arc()	ProGraphicsArcDraw()
progr_put_circle()	ProGraphicsCircleDraw()
progr_draw_polygon_2d()	ProGraphicsPolygonDraw()
Displaying Text	
progr_put_text()	ProGraphicsTextDisplay()
pro_get_text_attributes()	ProCurrentTextAttributesGet()
pro_set_text_attributes()	ProCurrentTextAttributesSet()
progr_get_default_font_id()	ProTextfontDefaultIdGet()
progr_get_font_id()	ProTextfontIdGet()
progr_get_font_name()	ProTextFontNameGet()
Getting Mouse Input	
promenu_get_pick()	ProMousePickGet()
pro_sample_xy()	ProMouseTrack()
progr_set_draw_mode()	ProGraphicsModeSet()
pro_getbox()	ProMouseBoxInput()
Display Lists	
pro_create_2d_disp_list()	ProDisplist2dCreate()
pro_display_2d_disp_list()	ProDisplist2dDisplay()
pro_delete_2d_disp_list()	ProDisplist2dDelete()
pro_create_3d_disp_list()	ProDisplist3dCreate()
pro_display_3d_disp_list()	ProDisplist3dDisplay()

pro_delete_3d_disp_list()	ProDisplist3dDelete()
Layers	
prolayer_get_names()	ProMdlLayerNamesGet()
prolayer_add_item()	ProLayerItemAdd()
prolayer_remove_item()	ProLayerItemRemove()
prolayer_display()	ProLayerDisplaystatusSet()
prolayer_get_display()	ProLayerDisplaystatusGet()
prolayer_create_layer()	ProLayerCreate()
prolayer_delete_layer()	ProLayerDelete()
prolayer_get_items()	ProLayerItemsGet()
Database Support	
Session Objects	
pro_get_current_object()	ProMdlCurrentGet()
pro_get_current_mode()	ProModeCurrentGet()
prodb_find_declared_objects()	ProMdlDeclaredDataList()
prodb_find_nobject_depend()	ProMdlDependenciesDataList()
prodb_get_object_info()	ProMdlMdlnameGet() ProMdlOriginGet() ProMdlExtensionGet() ProMdlDirectoryPathGet()
prodb_get_object_ptr()	ProMdlInit()
prodb_first_name_in_list()	ProSessionMdlList()
prodb_next_name_in_list()	ProSessionMdlList()
prodb_was_object_modified()	ProMdlModificationVerify()
pro_solid_id_from_ptr()	ProSolidToPostfixId()
pro_solid_ptr_from_id()	ProPostfixIdToSolid()
File Management Operations	
prodb_create_object()	ProSolidMdlnameCreate()
prodb_create_obj()	ProSolidMdlnameCreate()
prodb_retrieve_object()	ProMdlnameRetrieve()
prodb_save_object()	ProMdlSave()
prodb_rename_object()	ProMdlnameRename()
prodb_copy_object()	ProMdlnameCopy()
prodb_erase_object()	ProMdlErase()
prodb_backup_object()	ProMdlnameBackup()
Simplified Representations	
prodb_get_simplfd_rep_info()	ProSimprepActiveGet()
prodb_get_simplfd_rep_list()	ProSolidSimprepVisit()
prodb_retrieve_simplfd_rep()	ProAssemblySimprepRetrieve()
Selecting Objects	
pro_select()	ProSelect()
pro_get_selection()	ProSelect()

pro_set_and_get_selection()	ProSelect()
Tracing a Ray Through a Model	
pro_ray_x_model()	ProSolidRayIntersectionCompute()
Element Information	
pro_element_info()	PRO_IDENTIFY:Pro*IdGet(), Pro*Init()PRO_BELONG_TO: ProGeomitemFeatureGet()
Regenerating Models	
pro_regenerate()	ProSolidRegenerate()
pro_regenerate_object()	ProSolidRegenerate()
Part Accuracy	
prodb_get_model_accuracy()	ProSolidAccuracyGet()
prodb_set_model_accuracy()	ProSolidAccuracySet()
Mass Properties	
prodb_mass_prop()	ProSolidMassPropertyGet()
Utilities	
pro_str_to_wstr()	ProStringToWstring()
pro_wstr_to_str()	ProWstringToString()
pro_wchar_t_check()	ProWcharSizeVerify()
pro_show_file()	ProInfoWindowDisplay()
pro_show_info_window()	ProInfoWindowDisplay()
prodb_edit_file()	ProFileEdit()
Session Tools	
pro_get_prodevdat_info()	ProToolkitApplExecPathGet()
pro_getenvironment()	ProConfigoptGet()
pro_change_dir()	ProDirectoryChange()
pro_get_current_directory()	ProDirectoryCurrentGet()
pro_is_option_ordered()	ProOptionOrderedVerify()
pro_get_config()	ProConfigoptGet()
pro_set_config()	ProConfigoptSet()
progr_invalidate_display_list()	ProDisplistInvalidate()
Exporting and Importing Files	
pro_export_file_from_pro()	ProOutputFileMdlnameWrite()
pro_export_plot_file()	ProPlotfileWrite()
pro_read_file_to_pro()	ProInputFileRead()
pro_export_fea_mesh()	ProFemmeshExport()
Material Names	
prodb_get_material_props()	ProPartMaterialdataGet()
prodb_get_material_name()	ProPartMaterialNameGet()
prodb_set_material_name()	ProPartMaterialSet()
Storing Generic Data	
proappdata_register_class()	ProExtdataClassRegister()
proappdata_create_data()	ProExtdataSlotCreate()
proappdata_write_data()	ProExtdataSlotWrite()

proappdata_read_data()	ProExtdataSlotRead()
proappdata_delete_data()	ProExtdataSlotDelete()
proappdata_list_classes()	ProExtdataClassNamesList()
proappdata_list_data_in_class()	ProExtdataSlotIdsList()
Geometry	
Traversing the Geometry of a Part	
prodb_first_part_face()	ProSolidBodySurfaceVisit()
prodb_next_part_face()	ProSolidBodySurfaceVisit()
prodb_get_solid_surfaces()	ProSolidBodySurfaceVisit()
prodb_first_face_contour()	ProSurfaceContourVisit()
prodb_next_face_contour()	ProSolidBodySurfaceVisit()
prodb_first_cntr_edge()	ProContourEdgeVisit()
prodb_next_cntr_edge()	ProContourEdgeVisit()
prodb_get_datum_curves()	ProSolidFeatVisit(), ProFeatureGeomitemVisit()
prodb_get_datum_surfaces()	ProSolidQuiltVisit(), ProQuiltSurfaceVisit()
prodb_edge_data()	ProEdgeNeighborsGet()
prodb_edge_direction()	ProEdgeDirGet()
prodb_contour_traversal()	ProContourTraversalGet()
prodb_containing_contour()	ProContainingContourFind()
prodb_vertex_data()	ProEdgeVertexdataGet()
prodb_get_solid_volumes()	ProSldsurfaceVolumesFind()
Evaluating Geometry	
prodb_edge_tessellation()	ProEdgeTessellationGet()
pro_eval_xyz_edge()	ProEdgeXyzdataEval()
pro_eval_xyz_entity()	ProCurveXyzdataEval()
pro_eval_uv_edge()	ProEdgeUvdataEval()
pro_eval_xyz_face()	ProSurfaceXyzdataEval()
pro_get_edge_param()	ProEdgeParamEval()
pro_get_entity_param()	ProCurveParamEval()
pro_get_face_params()	ProSurfaceParamEval()
prodb_uv_in_face_domain()	ProSurfaceUvpntVerify()
pro_point_on_geom()	ProGeometryAtPointFind()
prodb_get_edge_uv_points()	ProEdgeTesselationGet()
prodb_get_surface_tessellation()	ProSurfaceTessellationGet()
Geometry Equations	
prodb_get_curve_type()	ProCurveTypeGet()
pro_get_curve_type_geom()	ProCurvedataGet()
prodb_get_edge_type()	ProEdgeTypeGet()
prodb_get_edge_curve()	ProEdgedataGet()
prodb_rls_edge_curve()	ProGeomitemdataFree()
prodb_get_face_type()	ProSurfaceTypeGet()

prodb_get_surface()	ProSurfacedataGet()
prodb_rls_surface()	ProGeomitemdataFree()
prodb_surface_to_nurbs()	ProSurfaceToNURBS()
prodb_edge_to_nurbs()	ProEdgeToNURBS()
prodb_entity_to_nurbs()	ProCurveToNURBS()
Measurement	
pro_edge_length()	ProEdgeLengthEval()
pro_face_area()	ProSurfaceAreaEval()
prodb_measure()	ProSurfaceDiameterEval(), ProGeomitemAngleEval(), ProGeomitemDistanceEval()
pro_face_extremes()	ProSurfaceExtremesEval()
prodb_get_envelope()	ProSolidOutlineGet()
prodb_compute_outline()	ProSolidOutlineCompute()
Parameters and Dimensions	
Parameters	
prodb_get_parameters()	ProParameterVisit() and ProParameterValueWithUnitsGet()
prodb_set_parameters()	ProParameterValueWithUnitsSet()
prodb_add_parameters()	ProParameterCreate()
prodb_delete_parameters()	ProParameterDelete()
prodb_reset_parameters()	ProParameterValueReset()
prodb_designate_param()	ProParameterDesignationAdd() ProParameterDesignationVerify()
Dimensions	
prodim_display_dimension	ProAnnotationDisplay()
prodim_get_dim_text	ProDimensionTextWstringsGet()
prodb_dim_is_visible	ProDimensionIsAccessibleInModel()
prodim_get_dimension	ProDimensionSymbolGet() ProDimensionValueGet() ProDimensionToleranceGet() ProDimensionTypeGet() ProDimensionIsFractional() ProDimensionDecimalsGet() ProDimensionDenominatorGet() ProDimensionIsReldriven() ProDimensionIsRegenednegative().
Features	
Listing Features	
prodb_get_feature_ids()	ProSolidFeatVisit()
prodb_get_feat_parent_child()	ProFeatureParentsGet()
prodb_get_feat_type()	ProFeatureTypeGet()

Names of Features and Other Elements	
prodb_get_element_name()	ProModelitemNameGet()
prodb_set_element_name()	ProModelitemNameSet()
Feature Geometry	
prodb_get_feature_surfaces()	ProFeatureGeomitemVisit()
prodb_get_surface_feature()	ProGeomitemFeatureGet()
Manipulating Features	
prodb_suppress_feature()	ProFeatureSuppress()
prodb_resume_feature()	ProFeatureResume()ProSolidFeatstatusSet()
prodb_delete_feature()	ProFeatureDelete()
Datum Planes	
prodb_first_datum()	ProSolidFeatVisit()
prodb_next_datum()	ProSolidFeatVisit()
Datum Points	
prodb_get_feature_dtm_points()	ProSolidFeatVisit()
prodb_eval_xyz_dtm_point()	ProPointCoordGet()
Axes	
prodb_get_first_axis()	ProSolidAxisVisit()
prodb_get_next_axis()	ProSolidAxisVisit()
Coordinate Systems	
prodb_find_csys()	ProSolidCsysVisit(),
prodb_unpack_csys()	ProCsysdataGet()
Surface Quilts	
prodb_get_surface_quilt()	ProSurfaceQuiltGet()
Surface Quilts	
prodb_get_surface_quilt()	ProSurfaceQuiltGet()
Assemblies	
Finding Assembly Members	
prodb_first_member()	ProSolidFeatVisit()
prodb_next_member()	ProSolidFeatVisit()
prodb_member_to_object()	ProAsmcompMdlGet()
prodb_obj_from_assem_rel()	ProPostfixIdToSolid()
Location of Assembly Members	
prodb_member_transform()	ProAsmcomppathTrfGet()
pro_vectors_to_transf()	ProMatrixInit()
prodb_get_asm_transform()	ProAsmcomppathTrfGet()
prodb_set_member_transform()	ProAsmcomppathTrfSet()
Assembling and Deleting Members	
prodb_assemble_by_transform()	ProFeatureCreate()
prodb_assemble_component()	ProFeatureCreate()
prodb_explode_assembly()	ProAssemblyExplode()
prodb_get_asm_constraints()	ProFeatureElemtreeCreate()
prodb_is_asm_exploded()	ProAssemblyIsExploded()

Notify	
Basic Notify Functions	
prodev_notify()	ProNotificationSet()
pro_on_regenerate_end()	ProNotificationSet()
Notify for File Management Operations	
user_menu_dbms_save_pre()	ProMdlSavePreAction()
user_menu_dbms_save_post()	ProMdlSavePostAction()
user_menu_dbms_copy_pre()	ProMdlCopyPreAction()
user_menu_dbms_copy_post()	ProMdlCopyPostAction()
user_menu_dbms_rename_pre()	ProMdlRenamePreAction()
user_menu_dbms_rename_post()	ProMdlRenamePostAction()
user_menu_dbms_erase_pre()	ProMdlErasePreAction()
user_menu_dbms_erase_post()	ProMdlErasePostAction()
user_menu_dbms_purge_pre()	ProMdlPurgePreAction()
user_menu_dbms_purge_post()	ProMdlPurgePostAction()
user_menu_dbms_delete_pre()	ProMdlDeletePreAction()
user_menu_dbms_delete_post()	ProMdlDeletePostAction()
user_menu_dbms_create_pre()	ProMdlCreatePreAction()
user_menu_dbms_create_post()	ProMdlCreatePostAction()
user_menu_dbms_retrieve_pre()	ProMdlRetrievePreAction()
user_menu_dbms_retrieve_post()	ProMdlRetrievePostAction()
user_dbms_save_post_all()	ProMdlSavePostAllAction()
user_dbms_copy_post_all()	ProMdlCopyPostAllAction()
user_dbms_erase_post_all()	ProMdlErasePostAllAction()
user_dbms_delete_post_all()	ProMdlDeletePostAllAction()
user_dbms_retrieve_post_all()	ProMdlRetrievePostAllAction()
Failure Notify	
user_dbms_failure_function()	ProMdlDbmsFailureAction()
Change Notify	
user_change_window_post()	ProWindowChangePostAction()
user_change_directory()	ProDirectoryChangePostAction()
Graphics Notify	
user_graphics_object_output_pre()	ProMdlDisplayPreAction()
user_graphics_object_output_post()	ProMdlDisplayPostAction()
Manufacturing Notify	
user_mfg_oper_cl_file_post()	ProMfgoperClPostAction()
user_mfg_feat_cl_file_post()	ProNcseqClPostAction()
Manufacturing Operations	
Manufacturing Components	
promfg_get_tool_ids() prodb_get_comp_role()	ProMfgAssemGet()

Manufacturing Parameters	
promfg_get_tool_ids()	ProMfgToolVisit()
promfg_get_tool_parameters()	ProToolParamGet()
promfg_set_tool_parameters()	ProToolElemParamAdd()
promfg_get_nc_type()	ProNcseqTypeGet()
Cabling Operations	
procbl_create_cables_from_logical()	ProCablesFromLogicalCreate()
procbl_get_cables_from_logical()	ProCablesFromLogicalGet()
procbl_get_connectors_from_logical()	ProConnectorsFromLogicalGet()
procbl_get_assy_spools()	ProAssemblySpoolsCollect()
procbl_create_spool()	ProSpoolCreate()
procbl_get_spool_params()	ProSpoolParametersCollect()
procbl_get_spool_param()	ProSpoolParameterGet()
procbl_get_assy_connectors()	ProAssemblyConnectorsGet()
procbl_designate_connector()	ProConnectorDesignate()
procbl_undesignate_connector()	ProConnectorUndesignate()
procbl_get_connector_entry_ports()	ProConnectorEntryPortsGet()
procbl_set_spool_params()	ProSpoolParametersSet()
procbl_delete_spool_param()	ProSpoolParameterDelete()
procbl_get_spools_from_logical()	ProSpoolsFromLogicalGet()
procbl_get_spools_from_logical()	ProSpoolsFromLogicalGet()
procbl_get_connector()	ProConnectorParamGet()
procbl_delete_connector_param()	ProConnectorParamDelete()
procbl_get_connector_params()	ProConnectorParamsCollect()
procbl_set_connector_params()	ProConnectorParamsSet()
procbl_get_assy_harnesses()	ProAssemblyHarnessesCollect()
procbl_create_harness()	ProHarnessCreate()
procbl_get_harness_cables() procbl_get_cable_id()	ProHarnessCablesCollect()
procbl_get_harness_locations()	ProHarnessLocationsCollect()
procbl_get_wire_harnesses()	ProCableHarnessesGet()
procbl_get_wire_length()	ProCableLengthGet()
procbl_get_cable_id()	ProCableByNameGet()
procbl_get_cable_name()	ProCableNameGet()
procbl_get_cable_subtype()	ProCableTypeGet()
procbl_create_cable()	ProCableCreate()
procbl_create_bundle()	ProBundleCreate()

procbl_get_bundle_cables()	ProBundleCablesCollect()
procbl_get_cable_logical_ends()	ProCableLogicalEndsGet()
procbl_get_cable_geom()	ProCableIsComplete()
procbl_set_cable_params()	ProCableParametersSet()
procbl_delete_cable_param()	ProCableParameterDelete()
procbl_get_cable_param()	ProCableParameterGet()
procbl_get_cable_params()	ProCableParametersCollect()
procbl_cable_clearance()	ProCableClearanceCompute()
procbl_get_cable_geom()	ProCableSegmentsGet()
procbl_get_cable_geom()	ProCablesegmentPointsGet()
procbl_get_cable_geom()	ProCablesegmentIsInBundle()
procbl_get_cable_geom()	ProCablesegmentIsNew()
procbl_get_cable_locations()	ProCableLocationsCollect()
procbl_get_location_cables() procbl_get_cable_id()	ProCablelocationCablesGet()
procbl_get_location_type()	ProCablelocationTypeGet()
procbl_get_location_pnt()	ProCablelocationPointGet()
procbl_routing_start()	ProCableRoutingStart()
procbl_route_thru_location()	ProCableThruLocationRoute()
procbl_routing_end()	ProCableRoutingEnd()
UDF Function	
prodb_first_udf()	ProSolidGroupVisit() or ProSolidGroupsCollect()
prodb_next_udf()	ProSolidGroupVisit() or ProSolidGroupsCollect()
prodb_get_group()	ProFeatureGroupGet() plusProGroupFeatureVisit() or ProGroupFeaturesCollect() plusProGroupIsTabledriven()
prodb_get_udf_name()	ProUdfNameGet()
prodb_first_dim_udf()	ProUdfDimensionVisit() or ProUdfDimensionsCollect()
prodb_next_dim_udf()	ProUdfDimensionVisit() or ProUdfDimensionsCollect()
prodb_get_udf_dim_name()	ProUdfDimensionNameGet()
prodb_place_udf()	Replaced
prodb_create_group()	ProUdfCreate()
prodb_get_udf_instance_name()	ProUdfNameGet()
Cross Sections	
prodb_create_parallel_xsec()	ProXsecParallelCreate()
prodb_delete_xsec()	ProXsecDelete()
prodb_display_xsec()	ProXsecDisplay()
prodb_first_xsec()	ProSolidXsecVisit()
prodb_mass_prop_xsec()	ProXsecMassPropertyCompute()
prodb_next_xsec()	ProSolidXsecVisit()

prodb_regen_xsec()	ProXsecRegenerate()
prodb_xsec_component()	ProXsecGeometryCollect()
prodb_xsec_name()	ProSolidXsecVisit()
Table-Driven Patterns	
proptntbl_add_inst_to_table()	Use the Table Pattern feature element tree documented in the header file ProPattern.h.
proptntbl_delete_table()	
proptntbl_get_active_table()	
proptntbl_get_all_tbl_names()	
proptntbl_get_inst_dim_ value()	
proptntbl_get_inst_indices()	
proptntbl_get_lead_pat_dims()	
proptntbl_remove_instance()	
proptntbl_rename_table()	
proptntbl_set_active_table()	
proptntbl_set_dimval_driven()	
proptntbl_set_inst_dim_ value()	
Merge and Cutout	
prodb_merge_members()	Use the Merge feature element tree documented in the header file ProMerge.h.
Automatic Interchange	
prodb_auto_interchange	ProAssemblyAutointerchange()
Drawing	
pro_get_drawing_text_height()	ProDrawingSetupOptionGet()
pro_set_drawing_text_height()	ProDrawingSetupOptionSet()
prodrw_dim_view_id()	ProDrawingDimensionViewGet()
prodrw_get_view_sheet()	ProDrawingViewSheetGet()
Notebook	
prodb_declare_layout()	ProLayoutDeclare()
prodb_undeclare_layout()	ProLayoutUndeclare()
prodb_regenerate_layout()	ProLayoutRegenerate()
Relations	
prodb_import_relations()	ProInputFileRead()
prodb_export_relations()	ProOutputFileMdlnameWrite()
Surface Properties	
prodb_get_surface_props	ProSurfaceAppearancepropsGet() or or ProSurfaceTexturepropsGet() or ProSurfaceTextureplacementpropsGet()
prodb_set_surface_props	ProSurfaceAppearancepropsSet() or ProSurfaceTexturepropsSet() or ProSurfaceTextureplacementpropsSet()
prodb_unset_surface_props	ProSurfaceAppearancepropsSet() or ProSurfaceTexturepropsSet() or ProSurfaceTextureplacementpropsSet()
Data structure pro_surf_props	Data structure ProSurfaceAppearanceProps,

	ProSurfaceTextureProps, and ProSurfaceTexturePlacementProps
pro_get_light_sources	ProLightSourcesGet()
pro_set_light_sources	ProLightSourcesSet()
Data structure Pro_light	ProLightInfo
prodb_surface_tessellation	ProSurfaceTessellationGet()
Interference	
pro_dist_manifolds()	ProSelectionDistanceEval() ProSelectionWithOptionsDistanceEval()
pro_compute_clearance()	ProFitClearanceCompute()
pro_compute_interference()	ProFitInterferenceCompute()
pro_compute_global_ interference()	ProFitGlobalinterferenceCompute()
pro_compute_volume()	ProFitInterferencevolumeCompute()
pro_display_interf_volume()	ProFitInterferencevolumeDisplay()
pro_interference_volume_ release()	ProInterferenceDataFree()
Customized Plot Driver	
prointerface_create() prointerface_load_function()	ProPlotdriverInterfaceCreate()
prointerface_object_set()	ProPlotdriverInterfaceobjectsSet()
prointerface_2d()	ProPlotdriverExecute()
user_intf_text()	ProPlotdriverTextPlot() ProPlotdriverTextfunctionSet()
user_intf_circle()	ProPlotdriverCirclePlot() ProPlotdriverCirclefunctionSet()
user_intf_arc()	ProPlotdriverArcPlot() ProPlotdriverArcfunctionSet()
user_intf_line()	ProPlotdriverLinePlot() ProPlotdriverLinefunctionSet()
user_intf_polyline()	ProPlotdriverPolylinePlot() ProPlotdriverPolylinefunctionSet()
user_intf_filled_poly()	ProPlotdriverPolygonPlot() ProPlotdriverPolygonfunctionSet()

The following Pro/Develop FEM functions have been replaced with equivalent Creo Parametric TOOLKIT functions in the mentioned header files:

Pro/DEVELOP Function	Equivalent Creo Parametric TOOLKIT Function
profem_get_con_case_names()	ProMechLoadset.h ProMechConstrset.h
profem_get_constraints()	ProMechLoad.h ProMechConstraint.h

Pro/DEVELOP Function	Equivalent Creo Parametric TOOLKIT Function
profem_get_mesh_controls()	ProMechMeshControl.h
profem_get_bar_elements()	ProMechBeam.h ProMechBeamOrient.h ProMechBeamRelease.h ProMechBeamSection.h ProMechSpring.h ProMechSpringProps.h ProMechGap.h ProMechWeld.h
profem_get_contacts()	ProMechContact.h
profem_get_mass_elements()	ProMechMass.h ProMechMassProps.h
profem_get_shell_pairs()	ProMechShellPair.h
pro_export_fea_mesh()	ProFemMesh.h with the function ProFemmeshExport()

J

Geometry Traversal

This appendix illustrates the relationships between faces, contours, and edges.

Overview

Note the following:

- A simple rectangular face has one contour and four edges.
- A contour will traverse a boundary such that the part face is always on the right-hand side (RHS). For an external contour, the direction of traversal is clockwise. For an internal contour, the direction of traversal is counterclockwise.
- If a part is extruded from a sketch that has a U-shaped cross section, there will be separate surfaces at each leg of the U-channel.
- If a part is extruded from a sketch that has a square-shaped cross section, and a slot feature is then cut into the part to make it look like a U-channel, there will be one surface across the legs of the U-channel. In other words, the original surface of the part is represented as one surface with a cut through it.
- Geometry traversal happens on body, solid surface and contour. For quilts, datum curves, edges, the geometry traversal will happen through a solid. To collect all the bodies in a specified solid, use the function `ProSolidBodiesCollect()`.

To Walk Through the Geometry of a Block

1. Walk through the surfaces of a body, using `ProSolidBodySurfaceVisit()`.
2. Walk through the surfaces of a solid, using `ProSolidSurfaceVisit()`.
3. Walk through the contours of each surface, using `ProSurfaceContourVisit()`.
4. Walk through the edges of each contour, using `ProContourEdgeVisit()`.

Geometry Terms

Consider the following definitions:

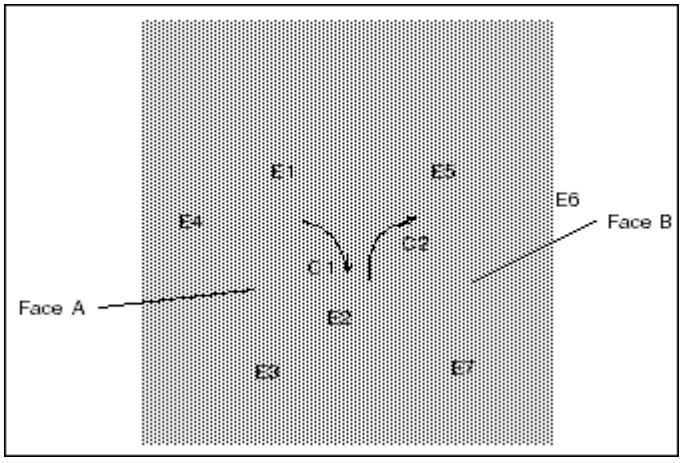
- surface—An ideal geometric representation, that is, an infinite plane.
- face—A trimmed surface. A face has one or more contours.
- contour—A closed loop on a face. A contour consists of multiple edges. A contour can belong to one face only.
- edge—The boundary of a trimmed surface.

  An edge of a solid is the intersection of two surfaces. The edge belongs to those two surfaces, hence to two contours. An edge of a datum surface can be

either the intersection of two datum surfaces, or the external boundary of the surface. If the edge is the intersection of two datum surfaces, it will belong to those two surfaces (hence, to two contours). If the edge is the external boundary of the datum surface, it will belong to that surface alone (hence, to a single contour).

Examples 1 through 5 show some sample parts and list the information about their surfaces, faces, contours, and edges.

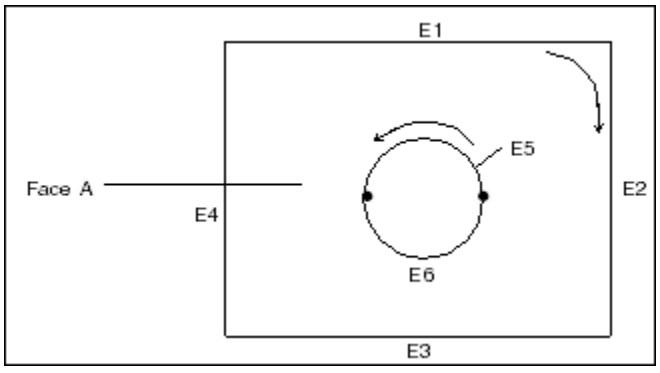
Example 1



This part has 6 faces.

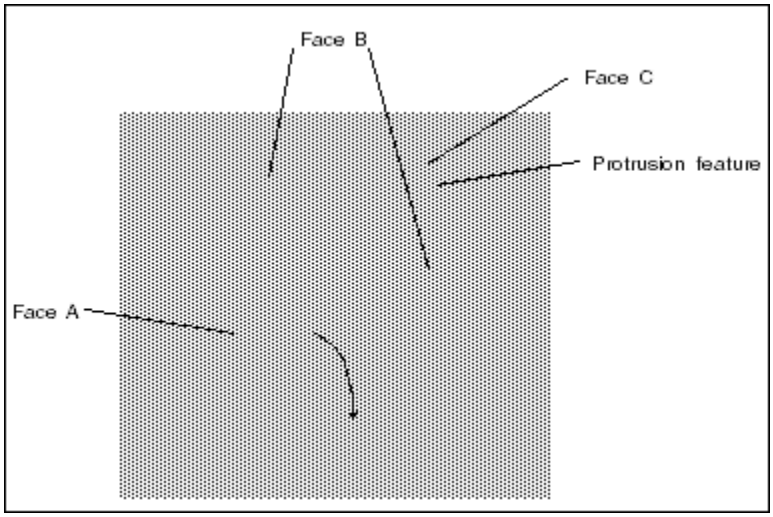
- Face A has 1 contour and 4 edges.
- Edge E2 is the intersection of faces A and B.
- Edge E2 is a component of contours C1 and C2.

Example 2



Face A has 2 contours and 6 edges.

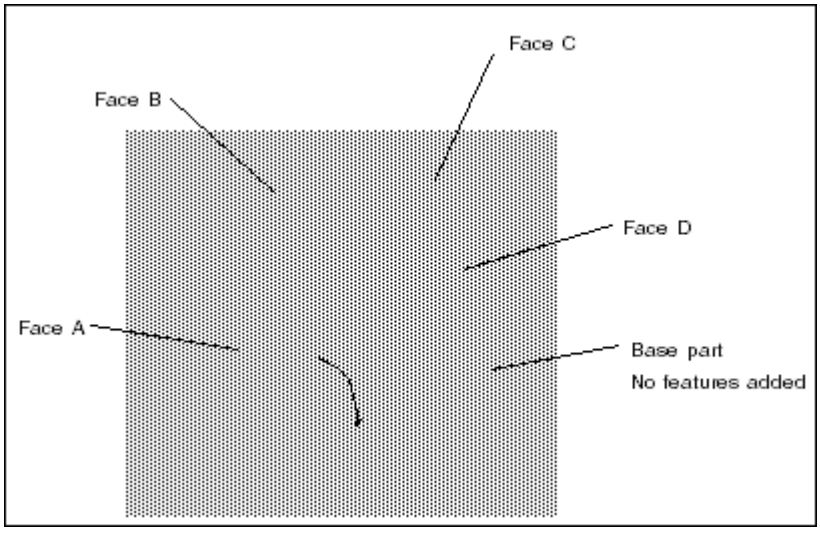
Example 3



This part was extruded from a rectangular cross section. The feature on the top was added later as an extruded protrusion in the shape of a semicircle.

- Face A has 1 contour and 6 edges.
- Face B has 2 contours and 8 edges.
- Face C has 1 contour and 4 edges.

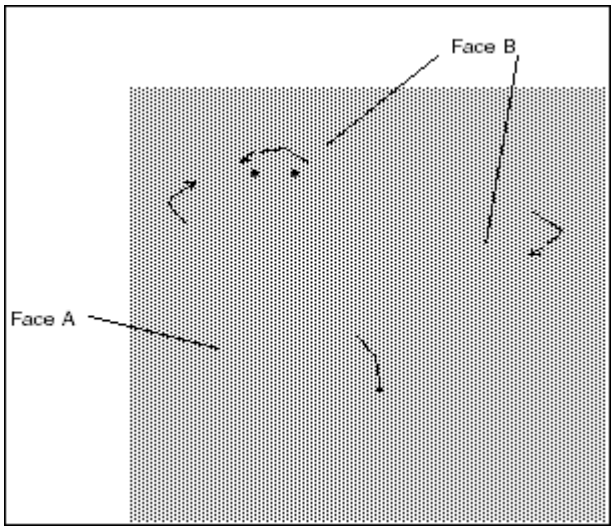
Example 4



This part was extruded from a cross section identical to Face A. In the Sketcher, the top boundary was sketched with two lines and an arc. The sketch was then extruded to form the base part, as shown.

- Face A has 1 contour and 6 edges.
- Face B has 1 contour and 4 edges.
- Face C has 1 contour and 4 edges.
- Face D has 1 contour and 4 edges.

Example 5



This part was extruded from a rectangular cross section. The slot and hole features were added later.

- Face A has 1 contour and 8 edges.
- Face B has 3 contours and 10 edges.

K

Geometry Representations

This appendix describes the geometry representations of the data structures defined in `ProGeomitem.h`. These structures are output by the geometry functions described in detail in the Core: 3D Geometry on page 170 chapter.

Domain of Evaluation

Surfaces and edges can be extended from their original domain as the user continues to add features to the model. For example, the user can add a feature such as a draft surface or local push, which requires the original surface to be extended outside its original domain.

When this occurs, you will find that the `uv` parameters of the surface have been extended. The `ProSurfacedata` data structure reflects the extension, and returns the updated values for the `u` and `v` extents.

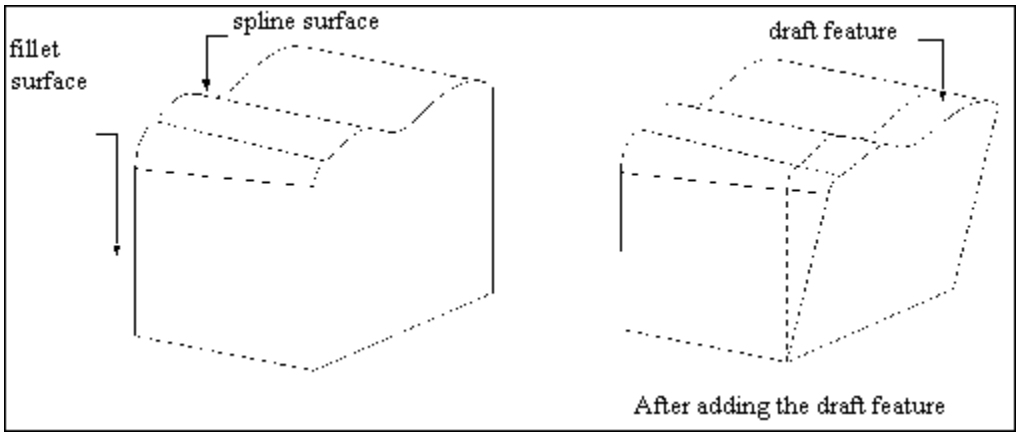
Because the evaluator functions use the analytical surface (or curve) definition, they work for any parameter values. Thus, any surface (or curve) can be extended as needed. In addition, if you pass in parameters outside the current `uv` domain, the evaluator functions still return values for the parameters as requested.

If you are using the evaluators supplied by Creo Parametric TOOLKIT, you do not have to do anything. For surfaces, the evaluator functions work over this extended range of parameters. Your evaluator function for foreign datum surfaces is also expected to allow for extrapolation.

Edges are always parameterized between `0.0` and `1.0`. When surfaces are extended, new edges are created that have parameters in the range `0.0` to `1.0`.

If you develop your own evaluator functions, you must be aware that the domain of a surface can be extended, as with foreign datum surfaces.

Surfaces



Surface Data Structures

The surface structure contains data that describes the boundary of the surface, and a pointer to the primitive surface on which it lies. The primitive surface is a three-dimensional geometric surface parameterized by two variables (`u` and `v`). The surface boundary consists of closed loops (contours) of edges. Each edge is attached to two surfaces, and each edge contains the `u` and `v` values of the portion

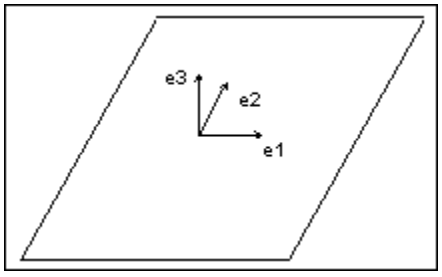
of the boundary that it forms for both surfaces. Surface boundaries are traversed clockwise around the outside of a surface, so an edge has a direction in each surface with respect to the direction of traversal.

Other data found in the surface structure includes the rectangular extents of the two-dimensional domain, the three-dimensional surface, and a flag indicating whether the surface normal points towards the inside or outside of the part. The user data is intended for run-time use only, and this information is not stored with the surface.

This section describes the surface data structures. The data structures are listed in order of complexity. For ease of use, the alphabetical listing of the data structures is as follows:

Plane

Plane



The plane entity consists of two perpendicular unit vectors (e1 and e2), the normal to the plane (e3), and the origin of the plane.

Data Format:

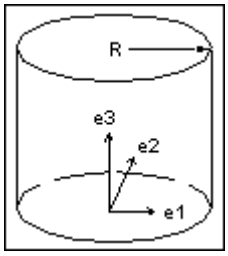
```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the plane
```

Parameterization:

```
(x, y, z) = u * e1 + v * e2 + origin
```

Cylinder

Cylinder



The generating curve of a cylinder is a line, parallel to the axis, at a distance R from the axis. The radial distance of a point is constant, and the height of the point isv.

Data Format:

```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the plane
radius      Radius of the cylinder
```

Parameterization:

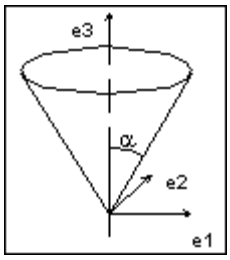
```
(x, y, z) = radius * [cos(u) * e1 + sin(u) * e2] +
          v * e3 + origin
```

Engineering Notes:

For the cylinder, cone, torus, and general surface of revolution, a local coordinate system is used that consists of three orthogonal unit vectors (e1, e2, and e3) and an origin. The curve lies in the plane of e1 and e3, and is rotated in the direction from e1 to e2. The u surface parameter determines the angle of rotation, and the v parameter determines the position of the point on the generating curve.

Cone

Cone



The generating curve of a cone is a line at an angle alpha to the axis of revolution that intersects the axis at the origin. The `v` parameter is the height of the point along the axis, and the radial distance of the point is `v * tan(alpha)`.

Data Format:

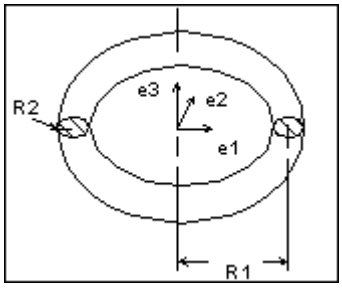
```
e1[3]      Unit vector, in the u direction
e2[3]      Unit vector, in the v direction
e3[3]      Normal to the plane
origin[3]  Origin of the cone
alpha      Angle between the axis of the cone
           and the generating line
```

Parameterization:

```
(x, y, z) = v * tan(alpha) * [cos(u) * e1 +
             sin(u) * e2] + v * e3 + origin
```

Torus

Torus



The generating curve of a torus is an arc of radius `R2` with its center at a distance `R1` from the origin. The starting point of the generating arc is located at a distance `R1 + R2` from the origin, in the direction of the first vector of the local coordinate system. The radial distance of a point on the torus is `R1 + R2 * cos(v)`, and the height of the point along the axis of revolution is `R2 * sin(v)` .

Data Format:

```
e1[3]      Unit vector, in the u direction
e2[3]      Unit vector, in the v direction
```

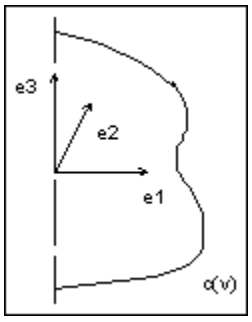
```
e3[3]       Normal to the plane
origin[3]   Origin of the torus
radius1     Distance from the center of the
            generating arc to the axis of
            revolution
radius2     Radius of the generating arc
```

Parameterization:

```
(x, y, z) = (R1 + R2 * cos(v)) * [cos(u) * e1 +
             sin(u) * e2] + R2 * sin(v) * e3 +
             origin
```

General Surface of Revolution

General Surface of Revolution



A general surface of revolution is created by rotating a curve entity, usually a spline, around an axis. The curve is evaluated at the normalized parameter `v`, and the resulting point is rotated around the axis through an angle `u`. The surface of revolution data structure consists of a local coordinate system and a curve structure.

Data Format:

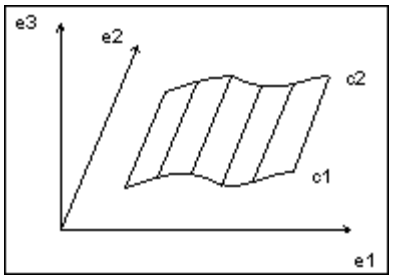
```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the surface of revolution
curve       Generating curve
```

Parameterization:

```
curve(v) = (c1, c2, c3) is a point on the curve.
(x, y, z) = [c1 * cos(u) - c2 * sin(u)] * e1 +
            [c1 * sin(u) + c2 * cos(u)] * e2 +
            c3 * e3 + origin
```

Ruled Surface

Ruled Surface



A ruled surface is the surface generated by interpolating linearly between corresponding points of two curve entities. The u coordinate is the normalized parameter at which both curves are evaluated, and the v coordinate is the linear parameter between the two points. The curves are not defined in the local coordinate system of the part, so the resulting point must be transformed by the local coordinate system of the surface.

Data Format:

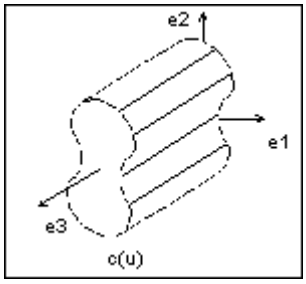
```
e1[3]      Unit vector, in the u direction
e2[3]      Unit vector, in the v direction
e3[3]      Normal to the plane
origin[3]  Origin of the ruled surface
curve_1    First generating curve
curve_2    Second generating curve
```

Parameterization:

```
(x', y', z') is the point in local coordinates.
(x', y', z') = (1 - v) * C1(u) + v * C2(u)
(x, y, z) = x' * e1 + y' * e2 + z' * e3 + origin
```

Tabulated Cylinder

Tabulated Cylinder



A tabulated cylinder is calculated by projecting a curve linearly through space. The curve is evaluated at the u parameter, and the z coordinate is offset by the v parameter. The resulting point is expressed in local coordinates and must be transformed by the local coordinate system to be expressed in part coordinates.

Data Format:

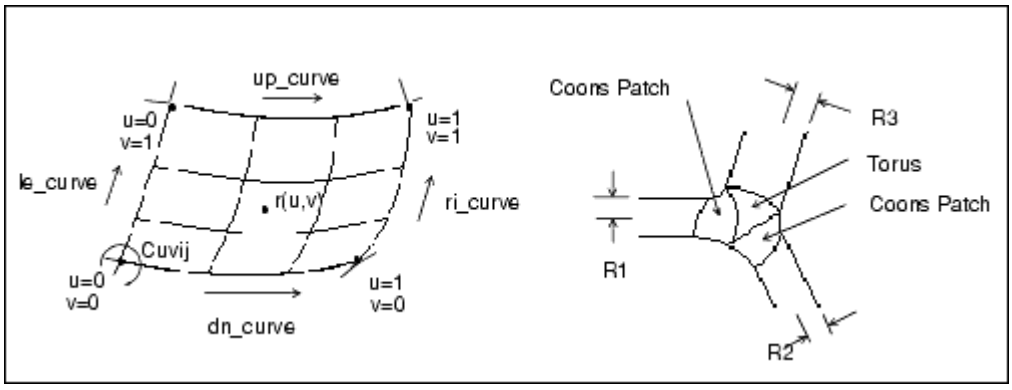
```
e1[3]      Unit vector, in the u direction
e2[3]      Unit vector, in the v direction
e3[3]      Normal to the plane
origin[3]  Origin of the tabulated cylinder
curve      Generating curve
```

Parameterization:

```
(x', y', z') is the point in local coordinates.
(x', y', z') = C(u) + (0, 0, v)
(x, y, z) = x' * e1 + y' * e2 + z' * e3 + origin
```

Coons Patch

Coons Patch



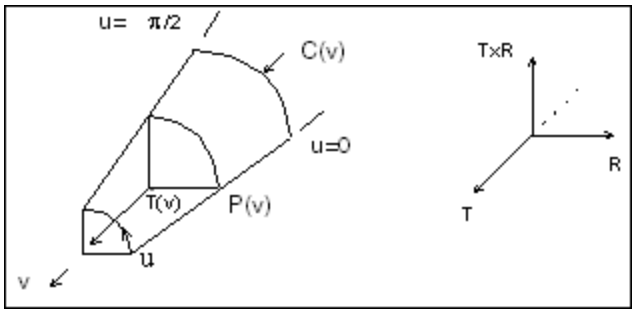
A Coons patch is used to blend surfaces together. For example, you would use a Coons patch at a corner where three fillets (each of a different radius) meet.

Data Format:

```
le_curve               u = 0 boundary
ri_curve               u = 1 boundary
dn_curve               v = 0 boundary
up_curve               v = 1 boundary
point_matrix[2][2]     Corner points
uvder_matrix[2][2]     Corner mixed derivatives
```

Fillet Surface

Fillet Surface



A fillet surface is found where a round or a fillet is placed on a curved edge, or on an edge with non-constant arc radii. On a straight edge, a cylinder would be used to represent the fillet.

Data Format:

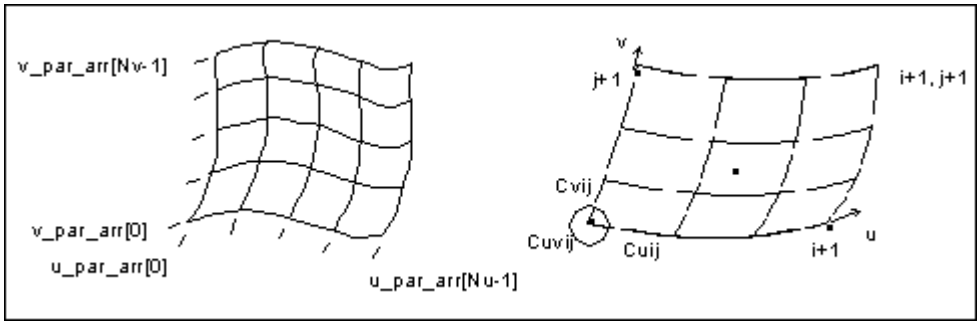
```
pnt_spline   P(v) spline running along the u = 0 boundary
ctr_spline   C(v) spline along the centers of the
             fillet arcs
tan_spline   T(v) spline of unit tangents to the
             axis of the fillet arcs
```

Parameterization:

```
R(v) = P(v) - C(v)
(x,y,z) = C(v) + R(v) * cos(u) + T(v) X R(v) *
          sin(u)
```

Spline Surface

Spline Surface



The parametric spline surface is a nonuniform bicubic spline surface that passes through a grid with tangent vectors given at each point. The grid is curvilinear in uv space. Use this for bicubic blending between corner points.

Data Format:

```
u_par_arr[]     Point parameters, in the u
```

```
                direction, of size Nu
v_par_arr[]     Point parameters, in the v
                direction, of size Nv
point_arr[][3]  Array of interpolant points, of
                size Nu x Nv
u_tan_arr[][3]  Array of u tangent vectors
                at interpolant points, of size
                Nu x Nv
v_tan_arr[][3]  Array of v tangent vectors at
                interpolant points, of size
                Nu x Nv
uvder_arr[][3]  Array of mixed derivatives at
                interpolant points, of size
                Nu x Nv
```

Engineering Notes:

- Allows for a unique 3x3 polynomial around every patch.
- There is second order continuity across patch boundaries.
- The point and tangent vectors represent the ordering of an array of `[i][j]`, where `u` varies with `i`, and `v` varies with `j`. In walking through the`point_arr[][3]`, you will find that the innermost variable representing `v(j)` varies first.

Second Derivative Spline Surface

The Second Derivative Spline Surface (`ProSpline2ndDersrfdata`) is a bicubic spline surface with possibility of altering the degree of boundary segments to accommodate corresponding curvature (2nd derivatives) conditions. Use this for bicubic blending with second degree derivatives along boundaries.

u_par_arr[]	Point parameters, in the u direction, of size Nu
v_par_arr[]	Point parameters, in the v direction, of size Nv
point_arr[][3]	Array of interpolant points, of size Nu x Nv
u_tan_arr[][3]	Array of u tangent vectors at interpolant points, of size Nu x Nv
v_tan_arr[][3]	Array of v tangent vectors at interpolant points, of size Nu x Nv
uvder_arr[][3]	Array of mixed derivatives at interpolant points, of size Nu x Nv
(*u_der2_arrs[2])[3]	2 Arrays of 2nd U derivatives along V boundaries interpolation points, size of Nv.
(*v_der2_arrs[2])[3]	2 Arrays of 3rd V derivatives along U boundaries interpolation points, size of Nu.
(*uuv_der[2])[3]	2 Arrays of uuv mixed derivatives along V boundaries interpolation points, size of Nv.
(*vvu_der[2])[3]	Arrays of vvu mixed derivatives along U boundaries interpolation points, size of Nv.
der4[4][3] UUVV	derivatives at the corners

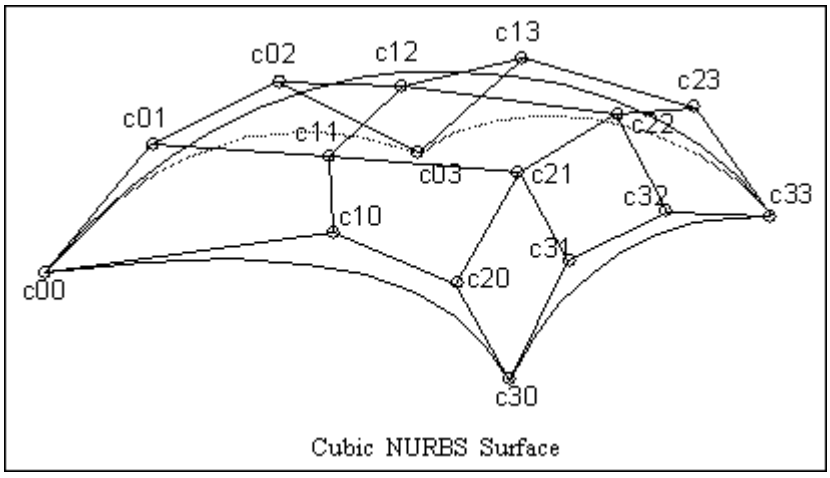
Engineering notes:

- Each second degree derivatives array can be `NULL`. If it is defined then corresponding 3rd degree mixed derivatives should be defined.
- Each der4 can be `NULL` and should be defined if 2nd degree derivatives are defined in both directions at the corresponding corner.

NURBS Surface

The NURBS (nonuniform rational B-spline) surface is defined by basis functions (in `u` and `v`), expandable arrays of knots, weights, and control points.

NURBS Surface



Cubic NURBS Surface

Data Format:

```
deg[2]              Degree of the basis
                    functions (in u and v)
u_par_arr[]         Array of knots on the
                    parameter line u
v_par_arr[]         Array of knots on the
                    parameter line v
wghts[]             Array of weights for
                    rational NURBS, otherwise
                    NULL
c_point_arr[][3]    Array of control points
```

Definition:

$$R(u,v) = \frac{\sum_{i=0}^{N1}\sum_{j=0}^{N2} C_{i,j} \times B_{i,k}(u) \times B_{j,l}(v)}{\sum_{i=0}^{N1}\sum_{j=0}^{N2} w_{i,j} \times B_{i,k}(u) \times B_{j,l}(v)}$$

```
k   = degree in u
l   = degree in v
N1  = (number of knots in u) - (degree in u) - 2
N2  = (number of knots in v) - (degree in v) - 2
B = basis function in u
B = basis function in v
w = weights
C = control points (x,y,z) * w
```

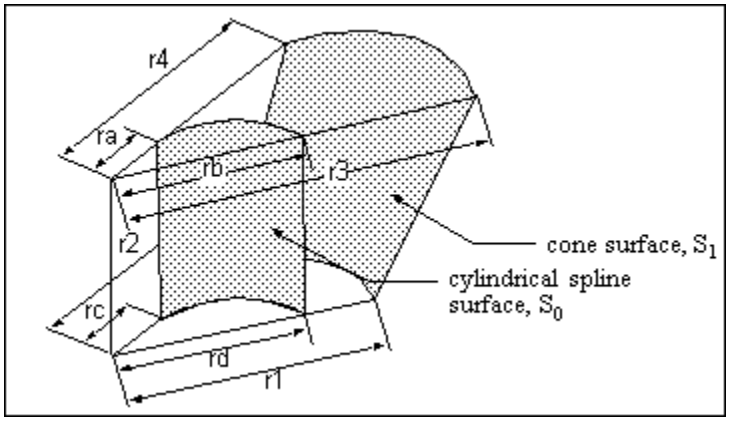
Engineering Notes:

The `weights` and `f` arrays represent matrices of size `wghts[N1+1][N2+1]` and `c_points_arr [N1+1] [N2+1]`. Elements of the matrices are packed into arrays in row-major order.

Cylindrical Spline Surface

The cylindrical spline surface is a nonuniform bicubic spline surface that passes through a grid with tangent vectors given at each point. The grid is curvilinear in modeling space.

Cylindrical Spline Surface



Data Format:

```
e1[3]      x' vector of the local coordinate system
e2[3]      y' vector of the local coordinate system
e3[3]      z' vector of the local coordinate system,
           which corresponds to the axis of revolution
           of the surface
origin[3]  Origin of the local coordinate system
splsrf     Spline surface data structure
```

The spline surface data structure contains the following fields:

```
u_par_arr[]         Point parameters, in the
                    u direction, of size Nu
v_par_arr[]         Point parameters, in the
                    v direction, of size Nv
```

```
point_arr[][3]      Array of points, in cylindrical
                    coordinates, of size Nu x Nv.
                     The array components are as follows:
                     point_arr[i][0] - Radius
                     point_arr[i][1] - Theta
                     point_arr[i][2] - Z
u_tan_arr[][3]       Array of u tangent vectors.
                     in cylindrical coordinates,
                     of size Nu x Nv
v_tan_arr[][3]      Array of v tangent vectors,
                    in cylindrical coordinates,
                    of size Nu x Nv
uvder_arr[][3]      Array of mixed derivatives,
                    in cylindrical coordinates,
                    of size Nu x Nv
```

Engineering Notes:

If the surface is represented in cylindrical coordinates (`r`,`theta`, `z`), the local coordinate system values (`x', y', z'`) are interpreted as follows:

```
x' = r cos (theta)
y' = r sin (theta)
z' = z
```

A cylindrical spline surface can be obtained, for example, by creating a smooth rotational blend (shown in the figure on the previous page).

In some cases, you can replace a cylindrical spline surface with a surface such as a plane, cylinder, or cone. For example, in the figure, the cylindrical spline surface S_1 was replaced with a cone

(`r1` = `r2`, and `r3` = `r4` and `r1` ≠ `r3`).

If a replacement cannot be done (such as for the surface S_0 in the figure (`ra` ≠ `rb` or `rc` ≠ `rd`)), leave it as a cylindrical spline surface representation.

Foreign Surface

The foreign surface consists of two perpendicular unit vectors (`e1` and `e2`), the normal to the plane (`e3`), the origin of the plane and the foreign ID returned by the function `user_init_surf()`.

Data Format:

```
e1[3]       Unit vector, in the u direction
e2[3]       Unit vector, in the v direction
e3[3]       Normal to the plane
origin[3]   Origin of the plane
foreign_id  Foreign ID returned by user_init_surf()
```

Parameterization is established by the Foreign Surface callback function `user_eval_surf()`.

Edge and Curve Data Structures

The data structures are used to represent edges (line, arc, and spline) as well as the curves (line, arc, spline, and NURBS) within the surface data structures.

This section describes the edge and curve data structures, arranged in order of complexity. For ease of use, the alphabetical listing of the edge and curve data structures is as follows:

Arc

The arc entity is defined by a plane in which the arc lies. The arc is centered at the origin, and is parameterized by the angle of rotation from the first plane unit vector in the direction of the second plane vector. The start and end angle parameters of the arc and the radius are also given. The direction of the arc is counterclockwise if the start angle is less than the end angle, otherwise it is clockwise.

Data Format:

```
vector1[3]     First vector that defines the
               plane of the arc
vector2[3]     Second vector that defines the
               plane of the arc
origin[3]      Origin that defines the plane
               of the arc
start_angle    Angular parameter of the starting
               point
end_angle      Angular parameter of the ending
               point
radius         Radius of the arc.
```

Parameterization:

```
t' (the unnormalized parameter) is
   (1 - t) * start_angle + t * end_angle
(x, y, z) = radius * [cos(t') * vector1 +
   sin(t') * vector2] + origin
```

Line

Data Format:

```
end1[3]   Starting point of the line
end2[3]   Ending point of the line
```

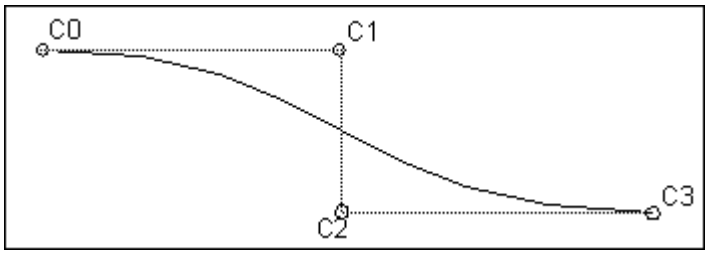
Parameterization:

```
(x, y, z) = (1 - t) * end1 + t * end2
```

NURBS

The NURBS (nonuniform rational B-spline) curve is defined by expandable arrays of knots, weights, and control points.

Cubic NURBS Curve



Data Format:

```
degree          Degree of the basis function
params[]        Array of knots
weights[]       Array of weights for rational
                NURBS, otherwise NULL.
c_pnts[][3]     Array of control points
```

Definition:

$$R(t) = \frac{\sum_{i=0}^{N} C_i \times B_{i,k}(t)}{\sum_{i=0}^{N} w_i \times B_{i,k}(t)}$$

`k` = degree of basis function

`N` = (number of knots) - (degree) - 2

`w` = weights

`C` = control points (x, y, z) * `w`

`B` = basis functions

By this equation, the number of control points equals `N+1`.

Spline

The spline curve entity is a nonuniform cubic spline, defined by a series of three-dimensional points, tangent vectors at each point, and an array of unnormalized spline parameters at each point.

Data Format:

```
par_arr[]     Array of spline parameters
              (t) at each point.
pnt_arr[][3]  Array of spline interpolant points
tan_arr[][3]  Array of tangent vectors at
              each point
```

Parameterization:

x, y, and z are a series of unique cubic functions, one per segment, fully determined by the starting and ending points, and tangents of each segment.

Let p_min be the parameter of the first spline point, and p_max be the parameter of the last spline point. Then, t', the unnormalized parameter, is t * p_max + (1-t) * p_min.

Locate the ith spline segment such that:

```
par_arr[i] < t' < par_arr[i+1]
```

(If t < 0 or t > +1, use the first or last segment.)

```
t0 = (t' - par_arr[i])   / (par_arr[i+1] - par_arr[i])
t1 = (par_arr[i+1] - t') / (par_arr[i+1] - par_arr[i])
The coordinates of the points are then:
(x, y, z) = pnt_arr[i] * t1^2 * (1 + 2 * t0) +
            pnt_arr[i+1] * t0^2 * (1 + 2 * t1) +
          (par_arr[i+1] - par_arr[i]) * t0 * t1 *
               (tan_arr[i] * t1 - tan_arr[i+1] * t0)
```

Ellipse

Ellipses in 3D geometry is split into two identical half-ellipses. The ellipse is defined by its major and minor axis radius values. Similar to arcs, elliptic segments are defined by a plane in which the ellipse lies, centered at the origin, and parameterized by the angle of rotation. The direction of the ellipse is counterclockwise if the start angle is less than the end angle, otherwise it is clockwise.

Data Format:

```
center  - Center point of the ellipse
major_axis_unit_vect  - Direction for the X-axis of the ellipse
norm_axis_unit_vect  - Direction for the Y-axis
major_len -  The “radius” in the X-direction
minor_leng - The “radius” in the Y-direction
start_ang - The ellipse start angle
end_ang - The end angle for the ellipse
```

The y-axis can be found as a vector product of norm_axis_unit_vect on the major_axis_unit_vect. In actual examples, the major_len can be less than the minor_len.

Parameterization:

```
t' (the unnormalized parameter) is
```

```
   (1 - t) * start_angle + t * end_angle
(x, y, z) = major_len * [cos(t') * vector1] +
   minor_len * [sin(t') * vector2] + origin
```

References:

Faux, I.D., M.J. Pratt. Computational Geometry for Design and Manufacture. Ellis Harwood Publishers, 1983.

Mortenson, M.E. Geometric Modeling. John Wiley & Sons, 1985.

L

Debugging Creo Parametric TOOLKIT Applications

This appendix describes how to debug Creo Parametric TOOLKIT applications.

Building a Creo Parametric TOOLKIT Application for Debugging

The following tables list commonly used changes for sample makefiles that allow debugging of a Creo Parametric TOOLKIT application. Consult your compiler documentation for more information on the flags specified below and other debugging options.

On Windows

Compile Line Changes	Link Line Changes
Change `CCFLAGS` or `CPPFLAGS` to add `/Od /Z7`.	Change the `/debug` flag to `/debug:full`. Remove or comment the lines that delete `$(OBJS)` after a successful build.

Debugging Techniques

Depending on the type and implementation for your application, you can use one of the following debugging techniques:

- If your application is a DLL that creates user interface and notification callbacks to be used by the interactive user, refer to the following section on Debugging an Interactive DLL.
- If your application is a DLL that runs in batch mode (from `user_initialize()`), refer to Debugging an Interactive DLL on page 2165.
- If your application is a spawn or asynchronous application, refer to Debugging a Multiprocess Application on page 2167.

Debugging an Interactive DLL

1. Start Creo Parametric with the DLL registered and loaded.
2. Use the debugger to attach to the process `xtop.exe`.
3. Set breakpoints in the callback functions that you wish to debug.

Debugging a Batch Mode DLL

Debugging Without Setting Breakpoints Before Loading Code

The following steps outline the procedure to debug applications using a debugger such as `dbx` that does not allow breakpoints to be set before the code is loaded.

1. Setup a Creo Parametric `start` command that uses the debugger as follows:
   a. Edit the `parametric.psf` file (or the related `psf.` file for the given command).

b. In this file change the RUN line for xtop, to invoke the debugger first, as:

```
RUN="$PRO_DIRECTORY/bin/parametric1"

RUN="dbx $PRO_DIRECTORY/bin/parametric1"
```

2. Run the Creo Parametric start command. The debugger starts.
3. Set up a breakpoint that triggers when the DLL is loaded.

   In dbx, enter stop dlopen <DLL name> For example, stop dlopen pt_inst_test.dll.

   If you are using a graphical debugger such as Sun's Workshop, the same procedure is possible. Workshop allows a "CUSTOM" breakpoint using dbx syntax, such as dlopenpt_inst_test.dll.
4. Enter run. The program runs to the point of loading the DLL.
5. When the dlopen breakpoint is triggered, add a breakpoint to user_initialize(). At this point, the DLL is loaded and the user_initialize() function is in memory.

Debugging With Breakpoints Set Before Loading Code

The following steps outline the procedure to debug applications using a debugger such as Microsoft Visual Studio that allows breakpoints to be set before the code is loaded.

1. Setup a Creo Parametric start command that uses the debugger as follows:
   a. Edit the parametric.psf file (or the related psf. file for the given command).
   b. In this file, change the RUN line for xtop, to invoke the debugger first as:

```
RUN="%PRO_DIRECTORY%\%PRO_MACHINE_TYPE%\obj\

xtop.exe"

RUN=devenv

"%PRO_DIRECTORY%\%PRO_MACHINE_TYPE%\obj\

xtop.exe"
```

2. Run the Creo Parametric start command. The debugger starts.
3. Load the source file containing the functions into the debugger, and graphically set a breakpoint to the start of user_initialize().
4. Run the program from the debugger User Interface. Even if the debugger indicates that no debugger information is available for xtop.exe, continue debugging.

When the DLL is loaded, the program will stop in the DLL's `user_initialize()`.

Debugging Creo Parametric TOOLKIT DLLs on Windows

The Creo Parametric executable contains a default exception handler on Windows. This handler will catch and react to exceptions generated by a Creo Parametric TOOLKIT DLL by default.

To allow debugging on Windows without using the Creo Parametric exception handler, set the environment variable `ALLOW_MS_DEBUG` to `true` in the `Control Panel`.

Setting `ALLOW_MS_DEBUG` to `true` forces Creo Parametric executable not to execute the exception handler and thus the process will stop in the debugger on any exception. Alternatively, it is possible to set exception handling in the debugger to stop on the first occurrence of the exception, even for exceptions that are handled.

Debugging a Multiprocess Application

Debugging a Synchronous Spawn Mode Application

1. Start Creo Parametric with the application registered and started.
2. Use the debugger to attach to the application process, for example `pt_inst_test.exe`.
3. Set breakpoints in the callback functions that you wish to debug.

Debugging an Asynchronous Spawn Mode Application

1. Start the application from the debugger. Set a breakpoint to stop the application before the call to `ProEngineerStart()` or `ProEngineerConnect()`.

Glossary

Creo Parametric TOOLKIT and Creo Parametric Terminology

This glossary contains words that have meanings specific to Creo Parametric TOOLKIT . For definitions of words that apply to Creo Parametric in general, see the Creo Parametric help.

Term	Definition
asynchronous mode	The mode in which Creo Parametric TOOLKIT can start or connect to a new Creo Parametric session and invoke operations in it.
body	The term “solid body” denotes a container object for solid geometry.
child	An item, such as view, part, or feature, that is dependent on another item for its existence. See also Parent.
complement mode	In this mode, the color becomes the exclusive or of the old color and the color being drawn. You can use this for creating dynamic graphics, such as rubberbands.
configuration file	A special text file that contains default settings for many Creo Parametric functions. Default environment, units, files, directories, and so on are set when Creo Parametric reads this file when it is started. A configuration file can reside in the startup directory to set the values for your working session only, or it can reside in the load directory to set values for all users running that version of Creo Parametric. Also known as the `config.pro` file.
contour	A closed loop on a face. A contour consists of multiple edges. A contour can belong to only one face.
coordinate system	A means of identifying points in space using a particular point in three-dimensional space (the origin) and three mutually perpendicular axes through the origin (the coordinate axes). Creo Parametric TOOLKIT uses four different coordinate systems: model, screen, window, and drawing.
curve	A continuous one-dimensional subset of three-dimensional space. The Creo Parametric TOOLKIT definition covers the geometry of not only datum curves, but also other features whose geometry is treated in the same way: axes and geometry edges.
display list	A list of vectors used to represent the shape of the model in the view.
domain	The portion or portions of a surface that correspond to real geometry.
draft entity	The graphical items created in Creo Parametric using the options under the **Sketch** tab. The possible values are arc, ellipse, line, point, polygon, spline, and so on.
draft group	A group of detail items that can contain notes and symbol instances, as well as draft entities.
edge	The curve along which two geometrical faces intersect.
element	A structural component in a Creo Parametric model that has its own internal identifier (and sometimes a name) and can be selected. Examples of elements are face, edge (but not contour), datum plane, datum surface, datum curve, axis, datum point, and feature.
entity	A geometric element that has the geometry of a curve, excluding edges.

Term	Definition
	Entities such as datum curves are accessed using their own Creo Parametric TOOLKIT functions, even though geometrically they behave like edges. Note that entities and draft entities are quite different: draft entities refer to two-dimensional items on a drawing.
evaluate	To invoke the evaluation, at a point on an edge or surface, of the parametric equations of Creo Parametric for that edge or surface. Evaluation provides a description of the three-dimensional geometry, in model coordinates.
face	A geometry element that describes a geometrical surface and its relationship with other geometry elements (edges and faces).
highlight	To emphasize an element by modifying its appearance on the workstation surface, usually by changing its color.
information window	A Creo Parametric window that displays information such as object lists, mass properties, and BOM.
leader	The arrow that points from a note or symbol to either a point on an edge of the geometry of a model in a drawing view or to a point in the drawing.
load directory	The directory where Creo Parametric is loaded.
macro keys	The function keys or key sequences for which you predefined a menu option or sequence of menu options. The predefined menu options enable you to pick a macro from a menu that is currently on the screen.
main menu	An autonomous menu with its own title. The other type of menu is a submenu.
mass properties	The information about the distribution of mass in the part or assembly. The C structure used to describe mass properties is `ProMassProperty`, and is declared in the header file `ProSolid.h`. The data structure includes fields for the following: volume, surface area, density, mass, center of gravity (COG), inertia matrix, inertia tensor, COG inertia tensor, eigenvalues of the COG inertia, and eigenvectors of the COG inertia.
matrix	A two-dimensional array used for transformations. See also Transformation.
menu	A list of options presented by Creo Parametric that you select using the mouse or predefined macro keys. See the User Interface: Menus, Commands, and Popupmenus on page 301 for more information on menus.
menu file	A text file that enables you to specify your own text for the name of a menu button, the one-line help text that appears when you place the cursor over that button, and translations for both of these.
message file	A text file that enables you to provide your own translation of the text message. The message file consists of groups of four lines, one group for each message that you want to write out.
model	A top-level object in a Creo Parametric mode.
model coordinates	The coordinate system in Creo Parametric is internally used to define the geometry of a model. You can visualize these coordinates by creating a coordinate system datum with the option **Model ► Coordinate System**.
model item	A generic object used to represent any item contained in any type of model, for the purpose of functions whose actions are applicable to all these types of item.
notify	Enables you to trap certain classes or events in the Creo Parametric session and arrange for a function in the Creo Parametric TOOLKIT program to be called before or after such a trapped event.
object	An item stored as a single file, such as a part, assembly, or drawing.
overlay view	The view for a whole drawing sheet.
parameter	A user-driven property that can be added to elements in a Creo Parametric

Term	Definition
	model and used to drive dimensional relations. Parameters consist of a name, type, and a value that can be an integer, double, or string. Parameters are accessible through the Creo Parametric user interface, as opposed to attributes that are private to Creo Parametric TOOLKIT .
parent	An item that has other items dependent upon it for their existence. For example, the base feature has all other features dependent upon it. If a parent is deleted, all dependent (children) items are deleted.
pipeline	A set of interconnecting pipes and fitments consisting of an extension which terminates at open ends, non-open ends, or junctions (branches).
pipeline branch	Pipes grouped into extensions such that the extension which continues across the branch has a continuous direction of flow.
pipeline extension	A non-branching sequence of pipeline items.
pipeline feature	A feature which names the pipeline to show its grouping but contains no geometry.
pipeline fitting	A component that connects two pipe segments, for example, a corner or a valve.
pipeline junction	An assembly component or a datum point that represents a part which joins three or more pipe segments.
pipeline member	A extension terminator, series, or junction.
pipeline network	A data structure which contains references to pipeline objects and are structured to show their connectivity and sequence in relation to the flow.
pipeline object	A segment, a fitting, or a stubin.
pipeline segment	A section of pipe, either straight or arced.
pipeline series	A non-branching sequence of pipeline objects.
pipeline stubin	A datum point which joints three or more series.
pipeline terminator	The open or non-open ends of the pipeline.
scene graph	A tree structure which consists of nodes. This term used in reference to the graphics data.
set mode	In this mode, any graphics draw command sets the appropriate pixels to the color being drawn.
submenu	A menu that acts as an extension to the menu above it. A submenu has no title and is active at the same time as the menu above it. Selecting from the menu above it does not close the submenu.
surface	A continuous two-dimensional subset of three-dimensional space.
synchronous mode	The normal mode of operation that makes it appear that Creo Parametric TOOLKIT is part of the Creo Parametric process.
tessellation	The process of subdividing an edge into multiple smaller edges.
transformation	A change from one coordinate system to another. A transformation between two coordinate systems is represented by a 4x4 matrix.
triangle strip	A strip of triangles that are connected to each other. The term is used in reference to the graphics data.
user attributes	The attributes added by a user to add description to the object beyond the geometric definition. For example, stock number, price, and cost per unit are all user attributes.
vector	A straight line segment that has both magnitude and direction.
version stamp	Provides a way of keeping track of changes in a Creo Parametric model to which your Creo Parametric TOOLKIT application may need to respond.
view	In Part mode, a view is the orientation of the object. In Drawing mode, a view

Term	Definition
	is part of the drawing that represents the model.
wide string	A data type that allows for the fact that some character sets (such as Japanese KANJI) use a bigger character set than can be coded into the usual 1-byte char type.
window	A rectangular area of the workstation surface in which you work or in which the system displays messages. Creo Parametric uses a Main Window, Message Window, and subwindows. You can also specify a single-window environment.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z