Pro/ENGINEER®

Wildfire™ 2.0

Pro/PROGRAM™
Help Topic Collection

Parametric Technology Corporation

6,545,671 B1 08-April-2003	5,140,321 18-August-1992	5,557,176 09-November-1996
GB2354685B 18-June-2003	5,423,023 05-June-1990	5,561,747 01-October-1996

Third-Party Trademarks

Adobe is a registered trademark of Adobe Systems. Advanced ClusterProven, ClusterProven, and the ClusterProven design are trademarks or registered trademarks of International Business Machines Corporation in the United States and other countries and are used under license. IBM Corporation does not warrant and is not responsible for the operation of this software product. AIX is a registered trademark of IBM Corporation. Allegro, Cadence, and Concept are registered trademarks of Cadence Design Systems, Inc. Apple, Mac, Mac OS, and Panther are trademarks or registered trademarks of Apple Computer, Inc. AutoCAD and Autodesk Inventor are registered trademarks of Autodesk, Inc. Baan is a registered trademark of Baan Company. CADAM and CATIA are registered trademarks of Dassault Systemes. COACH is a trademark of CADTRAIN, Inc. DOORS is a registered trademark of Telelogic AB. FLEX*lm* is a trademark of Macrovision Corporation. Geomagic is a registered trademark of Raindrop Geomagic, Inc. EVERSYNC, GROOVE, GROOVEFEST, GROOVE.NET, GROOVE NETWORKS, iGROOVE, PEERWARE, and the interlocking circles logo are trademarks of Groove Networks, Inc. Helix is a trademark of Microcadam, Inc. HOOPS is a trademark of Tech Soft America, Inc. HP-UX is a registered trademark and Tru64 is a trademark of the Hewlett-Packard Company. I-DEAS, Metaphase, Parasolid, SHERPA, Solid Edge, and Unigraphics are trademarks or registered trademarks of Electronic Data Systems Corporation (EDS). InstallShield is a registered trademark and service mark of InstallShield Software Corporation in the United States and/or other countries. Intel is a registered trademark of Intel Corporation. IRIX is a registered trademark of Silicon Graphics, Inc. LINUX is a registered trademark of Linus Torvalds. MatrixOne is a trademark of MatrixOne, Inc. Mentor Graphics and Board Station are registered trademarks and 3D Design, AMPLE, and Design Manager are trademarks of Mentor Graphics Corporation. MEDUSA and STHENO are trademarks of CAD Schroer GmbH. Microsoft, Microsoft Project, Windows, the Windows logo, Windows NT, Visual Basic, and the Visual Basic logo are registered trademarks of Microsoft Corporation in the United States and/or other countries. Netscape and the Netscape N and Ship's Wheel logos are registered trademarks of Netscape Communications Corporation in the U.S. and other countries. Oracle is a registered trademark of Oracle Corporation. OrbixWeb is a registered trademark of IONA Technologies PLC. PDGS is a registered trademark of Ford Motor Company. RAND is a trademark of RAND Worldwide. Rational Rose is a registered trademark of Rational Software Corporation. RetrievalWare is a registered trademark of Convera Corporation. RosettaNet is a trademark and Partner Interface Process and PIP are registered trademarks of “RosettaNet,” a nonprofit organization. SAP and R/3 are registered trademarks of SAP AG Germany. SolidWorks is a registered trademark of SolidWorks Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and in other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun logo, Solaris, UltraSPARC, Java and all Java based marks, and “The Network is the Computer” are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and in other countries. TIBCO, TIBCO Software, TIBCO ActiveEnterprise, TIBCO Designer, TIBCO Enterprise for JMS, TIBCO Rendezvous, TIBCO Turbo XML, TIBCO Business Works are the trademarks or registered trademarks of TIBCO Software Inc. in the United States and other countries. WebEx is a trademark of WebEx Communications, Inc.

Third-Party Technology Information

Certain PTC software products contain licensed third-party technology: Rational Rose 2000E is copyrighted software of Rational Software Corporation. RetrievalWare is copyrighted software of Convera Corporation. VisTools library is copyrighted software of Visual Kinematics, Inc. (VKI) containing confidential trade secret information belonging to VKI. HOOPS graphics system is a proprietary software product of, and is copyrighted by, Tech Soft America, Inc. G-POST is copyrighted software and a registered trademark of Intercim. VERICUT is copyrighted software and a registered trademark of CGTech. Pro/PLASTIC ADVISOR is powered by Moldflow technology. Moldflow is a registered trademark of Moldflow Corporation. The JPEG image output in the Pro/Web.Publish module is based in part on the work of the independent JPEG Group. DFORMD.DLL is copyrighted software from Compaq Computer Corporation and may not be distributed. METIS, developed by George Karypis and Vipin Kumar at the University of Minnesota, can be researched at http://www.cs.umn.edu/~karypis/metis. METIS is © 1997 Regents of the University of Minnesota. LightWork Libraries are copyrighted by LightWork Design 1990–2001. Visual Basic for Applications and Internet Explorer is copyrighted software of Microsoft Corporation. Parasolid © Electronic Data

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494 USA

Table Of Contents

Pro/PROGRAM Basics

About Pro/PROGRAM

Each model in Pro/ENGINEER contains a listing of major design steps and parameters that can be edited to work as a program. By running the program, you change the model according to new design specifications.

To enter the Pro/PROGRAM environment, click **Tools > Program** from the **PART** or **ASSEMBLY** menu.

The WHICH DESIGN Menu

Initially, you can gain access to only a design listing that exists in the model. However, whenever you edit a listing, a file is created that contains the latest design specifications. At this point, two design listings exist for the same model, **From Model** and **From File**. After you successfully incorporate design changes in the model, **From File** is deleted, and only **From Model** is available.

In those cases where a **From File** design listing exists, the **WHICH DESIGN** menu displays two commands:

- **From Model**—Retrieves a design listing built in the model.
- **From File**—Retrieves a design for a model from an existing file named `assemblyname.als` or `partname.pls`.

  Note: From Model reflects the current state of the model, while **From File** includes all new instructions that you have added during the last editing session.

To View the Model Design

1. From the **PART** or **ASSEMBLY** menu, click **Program**.
2. Click **Show Design** or **Edit Design** from the **PROGRAM** menu to view the model design.
   - If you choose **Show Design**, the program appears in an information window.
   - If you choose **Edit Design**, the program appears under the system editor—usually in the startup window.

   Note: In the header of every design listing, a `REVNUM` indicates the last model revision. The system uses this to detect if the design is outdated.

A typical design listing may contain any of the following:

- Input variables with their current values
- Relations
- IF-ELSE clauses

- Lists of all the features, parts, or assemblies in the design, which, when enclosed within "IF condition... ELSE... END IF" statements, create alternate design branches
- EXECUTE statements (Assembly mode only)
- INTERACT statements
- Feature suppression and order
- MASSPROP statement

Example: A Model Design

A listing for part CLAMP may look like this:

```
VERSION D-02-03
REVNUM 182
LISTING FOR PART CLAMP
INPUT
END INPUT
RELATIONS
d0 = d6 * 2
END RELATIONS
ADD FEATURE (initial number 1)
INTERNAL ID 1
TYPE = FIRST FEATURE
FORM = EXTRUDED
SECTION NAME = S2DOO2
DEPTH = BLIND
FEATURE'S DIMENSIONS:
D0 = 1.0
D1 = 2.4
....
D5 = 45.0
END ADD
```

Incorporating Changes

To Incorporate Changes in the Model

After you finish editing a Pro/PROGRAM listing, the system asks you if you want to incorporate the changes. To proceed, enter `Y`. If you enter `N`, the program is not executed.

If you want to run the program at any point, open a listing using the **Edit Design** command. Exiting the editor (no changes need to be made) starts the program. You are prompted to specify whether you want to incorporate the changes in the model.

In order to incorporate the changes in the model, the system may prompt you to enter variables.

> **Note:** After changes are incorporated in the model, a design file is deleted; only **From Model** is available for viewing, editing, or executing.

Entering Input Variables

When a model design has input variables, the system prompts you to enter their values whenever you regenerate the model or incorporate new instructions in the model. You can enter data using the following commands on the **GET INPUT** menu:

- **Current Vals**—When you run the program, it uses the current values without requesting your input.

  Note: If you want to check the current parameter values, choose **Show Design > From Model**. The information window displays the listing with the input variables and values assigned to them in the current model design.

- **Enter**—Enter new input values as prompted. Check boxes in the **INPUT SEL** menu control parameter selection. Pro/PROGRAM only prompts you to enter a new value for the checked parameters.

- **Read File**—When running a program, the system uses input from a file. Type a file name (including the extension, if any).

To Select or Modify Input Parameters

1. Click **PART > Program**, and then click **PROGRAM > Edit Design**. The system editor displays the current program for the model.
2. Edit the program input list.
3. Incorporate your changes into the model.
4. Click **GET INPUT > Enter**.
5. In the **INPUT SEL** menu, click the check boxes next to the input parameters for which you want to enter values, and then click **Done Sel**.

6. Enter the values as prompted in the message area.
7. Click **PROGRAM > Done/Return**.

Input from a File

Instead of entering variables manually, you can enter them from a file located in the current directory using **Read File**. The input file must have one input per line, formatted as follows:

```
param_name = value or expression
```

For example:

```
THICKNESS = 2.5
INCLUDE_VALVE = YES
MATERIAL = "STEEL"
```

If you enter parameters from a file that contains fewer parameters than are called for in the `INPUT` statement, the system assumes current values for the missing parameter.

If, on the contrary, the output file contains more variables than are needed for the execution, those parameters not pertaining to the program are disregarded.

Because the program ignores those parameters that do not pertain to this particular program, you can create an input file that acts as a global source for a number of models.

> **Note:** The system is case-sensitive when parameters and their values are read in from a file. Be consistent in specifying variables.

Execution Errors

About Execution Errors

When execution errors are encountered, the system reacts as follows:

In Part mode and Assembly mode:

- If the failure is due to a feature error (for example, if a dimension violates a Relations constraint), the information window opens with the description of the error, which is also written to the file `errors.lst.n`. You can then edit the design **From File** (to correct the error) or **From Model** (to start afresh).
- If the failure is due to a geometry error, Pro/ENGINEER enters a special error resolution environment called the Resolve environment, which has various functions to help you diagnose and resolve the error.

In Assembly mode:

If the failure occurs during assembly (for example, because a substituted member does not fit), the system informs you that it failed to replace the particular member and asks you if you want to reedit the program.

Feature Errors

Many errors are not detected during editing, but they still make the design unusable. They can be defined generally as invalid feature-list errors. Such errors usually result from reordering or deleting features that depend on each other, or from imposing condition values on features such that a feature that must be created is missing its parent.

Feature list errors are caught during execution, after the input values have been requested, but before the model geometry reconstruction has begun.

Geometry Errors

Some errors cannot be detected until the geometry reconstruction process has begun. For example, you could take the following design:

```
ADD FEATURE PROTRUSION
ADD FEATURE SHELL
OF THICKNESS d10 (to make a cup)
ADD FEATURE PROTRUSION (handle for cup)
```

and reorder the last two features, which are not dependent on each other. If the geometry of the handle is too thin to be shelled with thickness d10, this creates a geometry error. Pro/ENGINEER fails to regenerate the shell. The failed feature (the shell) is highlighted in red on the model. The **RESOLVE FEAT** menu appears, and the **Failure Diagnostics** window opens with information on the failed item.

If you choose **Undo Changes**, the system undoes the changes in this regeneration and returns to the previous state. In the preceding example, this means that the feature order would be restored.

Creating Instances

To Create a Part or Assembly Instance Programmatically

Parts and assemblies created programmatically with input parameters can be turned into instances of the generic model.

Whenever a design has been executed, either after regenerating the model or after editing the design, you can create a family instance of that specific configuration using the **Instantiate** command on the **PROGRAM** menu.

1. Click **PART > Program** or **ASSEMBLY > Program**.
2. On the **PROGRAM** menu, click **Instantiate**. Pro/TABLE appears with the name of the generic model or models (part in Part mode, assemblies and parts in Assembly mode) in column 1, and the default instance name or names in column 2.
3. Edit the default instance name or names for assembly and parts if desired, and then exit Pro/TABLE.

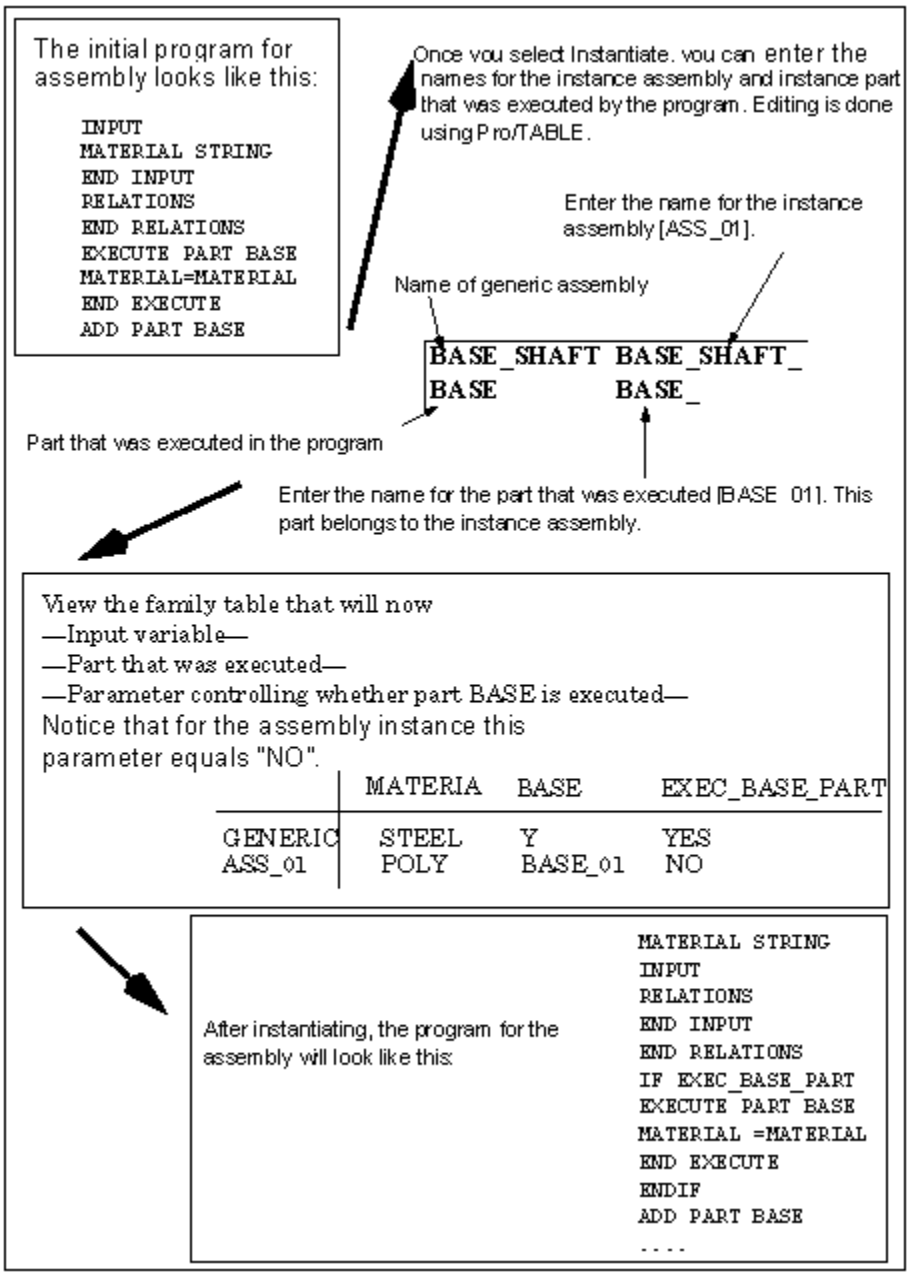
After you create an instance assembly, you can view the family table. It now includes the instance assembly name, part names that were executed, and variables that were entered during input.

Note: The parameters that appear in the family table control the model design.

Instantiating a model revises your design program slightly. For example, if an assembly program had an `EXECUTE` statement, an `IF` statement is created about the `EXECUTE` statement. This validates execution only for a generic assembly.

Example: Creating an Assembly Instance

An example of the typical workflow involved in creating an assembly instance follows:

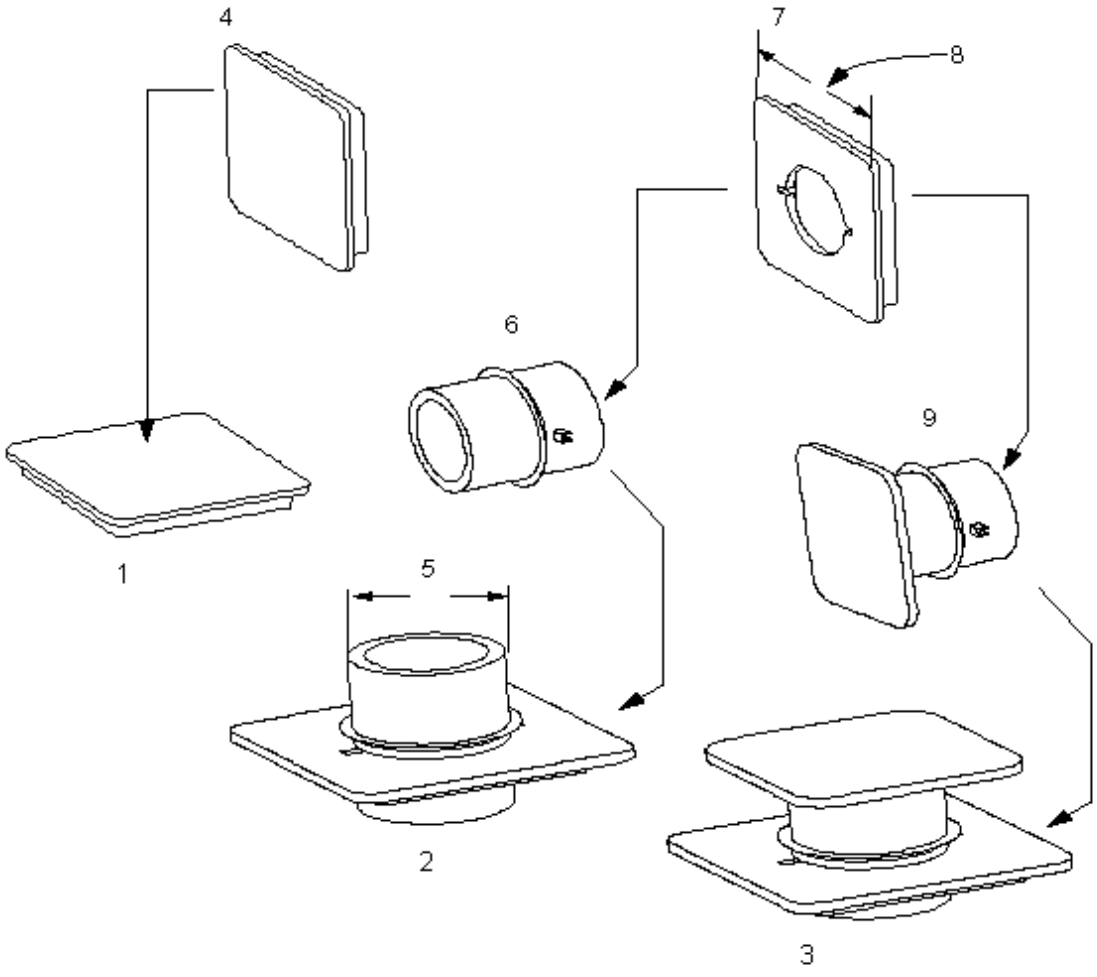


An Example of Parametric Design

Example: A Parametric Design for a Blender Cover

This example illustrates the logic of the design and the usage of `INPUT`, `EXECUTE`, and `IF-ELSE` statements. The format of the `ADD FEATURE` statements in the part design has been simplified. The explanations in square brackets are for information only and do not appear in a normal listing.

Creating a Parametric Design



1. Assembly 1
2. Assembly 2
3. Assembly 3
4. COVER_TYPE=NO
5. CYL_DIAM

6. CAP: MODEL_A
7. COVER_TYPE=YES
8. COVER_SIZE
9. CAP: MODEL_B

Design for Assembly BLENDER

The parametric design for the assembly BLENDER follows:

```
INPUT
COVER_TYPE YES_NO
"Does the cover have a cap?:"
MATERIAL   STRING
"Enter material (ABS or Poly):"
CAP_TYPE   STRING
"Enter cap type (MODEL_A or MODEL_B):"
COVER_SIZE
"Enter the top plate dimension:"
END INPUT
RELATIONS
END RELATIONS

EXECUTE PART COVER   [a.]
COVER_TYPE = COVER_TYPE
COVER_SIZE = COVER_SIZE
MATERIAL = MATERIAL
END EXECUTE

ADD PART COVER    [b.]
INTERNAL MEMBER ID 2
...
END ADD

IF COVER_TYPE == YES   [c.]
ADD PART (CAP_TYPE)
INTERNAL MEMBER ID 3
...
END ADD
END IF
```

Note:

a. Pass value for `COVER_TYPE` down to part "Cover." If value is `YES`, cover has a hole added. Also, pass values for material and size of the cover (size of the top plate).

b. Add a cover.

c. If `COVER_TYPE=YES`, add the cap to the assembly

Design for Part COVER

The parametric design file for the part COVER follows:

```
INPUT
COVER_TYPE YES_NO
COVER_SIZE
MATERIAL STRING
END INPUT

RELATIONS
DIAM = COVER_SIZE / 2   [a.]
IF MATERIAL == "Poly"
d0=.10
ELSE
d0=.2
ENDIF
END RELATIONS

ADD FEATURE 1
INTERNAL FEATURE ID 33   [b.]
TYPE=FIRST FEATURE
...
COVER_SIZE =   2.4   [c.]

ADD
END

ADD FEATURE   2    [d.]
INTERNAL FEATURE ID 169
TYPE=PROTRUSION
...
END ADD

IF COVER_TYPE == YES   [e.]
ADD FEATURE 3
INTERNAL FEATURE ID 270
TYPE=SLOT
...
END ADD
END IF
```

Note:

a. Relations include a relation for the hole diameter and a conditional statement for material type. ("Poly" and "ABS" require double quotation marks.)

b. Add the base feature.

c. Parameter name has been renamed to "COVER_SIZE".

d. Add walls.

e. If COVER_TYPE=YES, add a hole. (No quotation marks around YES.)

Design for Part CAP

The part CAP is table driven with instances MODEL_A and MODEL_B. The parametric design file for the part CAP follows:

```
INPUT
END INPUT

RELATIONS
END RELATIONS

ADD FEATURE 1   [Add the base feature of the cap.]
INTERNAL FEATURE ID 33
TYPE=FIRST FEATURE
...
END ADD

ADD FEATURE   2   [Add a datum plane.]
INTERNAL FEATURE ID 106
TYPE=DATUM PLANE
...
END ADD

ADD FEATURE   3   [Add a protrusion.]
INTERNAL FEATURE ID 108
TYPE=PROTRUSION
...
END ADD

ADD FEATURE   4   [Add a hole.]
INTERNAL FEATURE ID 179
TYPE=HOLE
...
END ADD

ADD FEATURE   5   [Add a top plate.]
INTERNAL FEATURE ID 198
TYPE=PROTRUSION
END ADD
```

Editing a Design

About Editing the Model Design

By editing a design, you make changes to a model.

Although the editor permits you to make other changes (changes not discussed in the following topic) in the design, it ignores these changes upon execution. Only those discussed in the following topic are actually changed in the design.

For example, if a feature attribute was changed from THRU ALL to THRU NEXT, the attribute that appears in the model after execution is THRU ALL.

To edit a design, click **PROGRAM > Edit Design**. If two designs exist for the model, you must choose **From Model** or **From File** from the **WHICH DESIGN** menu.

> **Note:** When you edit your design for the first time, or after you have successfully incorporated changes in the model, the **WHICH DESIGN** menu does not appear. In these cases, the design is edited only **From Model.**

A warning appears when you attempt to edit **From Model** while a file with a Pro/PROGRAM listing exists in the working directory. This warning reminds you that when you exit from the editor the file will be overwritten with the new contents. If you still want to proceed with editing (this replaces an old design file), enter `Y`. To abort editing, enter `N`.

If you are working with an assembly that has components belonging to a family table, listings for instances may be viewed, but they cannot be edited, because the program always resides in the generic part.

> **Note:** To gain access to the listing in a generic assembly, assembly instances must be cleared from workstation memory.

Relations

All relations valid in a Pro/ENGINEER model can be entered in a Pro/PROGRAM design.

If an expression you want to include in the RELATIONS statement contains more than 80 characters, use a backslash (\) to interrupt the current line and continue the expression on the next line.

The format can be as follows:

```
RELATIONS
PARAMETER = COVER_SIZE/2 + LENGTH*0.75 -\
0.75*d3*d3 + THICKNESS*2
END RELATIONS
```

Changing the material density in a part causes the system to update the `mp_density` value in relations and vice versa.

Note:

- When using negative dimensions, a dollar sign ($) must precede the dimension symbol in both the input statement and the external input files. For example, use `$d20` instead of `d20`. The dimensions will not be updated if a dollar sign does not precede the symbols.
- If the program assigns a value to a dimension variable that is already driven by a part or subassembly relation, two error messages appear. Edit or remove the program relation and regenerate.

Using Comments to Annotate Relations and Features

You can use comments in the program to annotate relations and features. To insert comments, use the following format:

```
/* < your comment
```

Note that the slash and asterisk precede the comment. Also, the comment on a feature must immediately follow its `ADD FEATURE` line. The comment is then attached to the feature being added and appears in the information window.

Input Parameters and Prompts

About Input Parameters and Prompts

INPUT variables may be specified at the beginning of the listing. A typical use of an INPUT variable is to supply a value for a dimension. This is a parameter later used in a relation or as input for model names used in assemblies.

Input Parameter Types

The INPUT statement must indicate the name and type of the variable. Variable names must always begin with a character.

The following variable types are supported:

- **Number**—Enter a number for this variable type.
- **String**—Enter a string of characters for this variable type. This enables you to enter parameters or model names, but not user attributes.
- **YES_NO**—Enter either `Y` or `N` for this variable type.

  Note: If no type is specified for the variable, the system default is `Number`.

Customizing Prompts for Input Variables

Whenever input is required, the system prompts you to enter the value of the input variable. Instead of using the system prompts, you can customize prompts for

particular input variables. Then, during design execution, the prompts appear when the associated variable requires input.

The rules for including prompts follow:

- A prompt must be enclosed in quotation marks.
- A prompt must immediately follow the corresponding input variable.

For example:

```
INPUT
THICKNESS NUMBER
"Enter wall thickness for the cylinder"
INCLUDE_VALVE YES_NO
"Is valve to be included for analysis"
STOCK_ID STRING
"Enter the part's stock ID"
...
END INPUT
```

Deleting Input Lines

If an input variable is deleted from the design or its name is changed, the relations and conditions that use it do not become invalid automatically. The old variable name remains in the list of parameters of the model and needs to be deleted explicitly using the **Del Param** command.

Conditional Input Statements

The input list in Pro/PROGRAM can include `IF - ELSE - ENDIF` statements. When an `IF` condition evaluates to `FALSE`, you are not prompted to enter the input values.

For example:

```
INPUT
INCLUDE_HOLE YES_NO
"Should the hole be included?:"
IF INCLUDE_HOLE == YES
HOLE_DIA NUMBER
"Enter diameter for hole"
ELSE
...
ENDIF
...
END INPUT
```

When executing this program, you are prompted to enter the diameter of a hole only if a hole feature is included.

IF-ELSE Statements

About Design Branches

Conditional statements can be used to create a design branch, enabling you to control whether a feature or component is included in the design.

For example, if the original Part design was:

```
ADD PROTRUSION.....
ADD HOLE.....
ADD CUT.....
```

The modified design might look like this:

```
ADD PROTRUSION.....
IF d1 > d2
ADD HOLE
...
END ADD
ENDIF
ADD CUT.....
END ADD
```

Conditional statements are also valid for assemblies. They control whether a particular part or subassembly is added to the assembly or executed. In the following example, `PART_B` is not used unless the parameter `DIA` has a value less than or equal to `1.25`.

```
ADD PART BASE_1
....
IF DIA > 1.25
ADD PART PART_A
.....
END ADD
ELSE
ADD PART PART_B
.....
END ADD
ENDIF
```

Pro/ENGINEER reevaluates any Pro/PROGRAM feature conditional statements (for example, `IF` statements) before regenerating each feature. As a result, only a single **Regenerate** command is needed for a design in which Pro/PROGRAM feature conditional statements are changed by Evaluate features and reference dimensions.

However, if you add to a design a Pro/PROGRAM feature conditional statement that is changed by a later feature, the system provides an error message that the design is now inconsistent.

Other Variable Types in IF Statements

All variable types may be included in IF statements. Notice that string values must be enclosed in quotation marks.

For strings:

```
IF MATERIAL == "STEEL"
d2=10
ENDIF
```

For YES_NO:

```
IF DRAFT==YES
d25=5
ENDIF
```

Replacing Components in Assembly Designs

About Replacing Components in Assembly Designs

You can set up a program to replace assembly components with interchangeable components. Interchangeability is established using interchange groups, members of the same family table, or assembly layout declarations. The member named when executing the design must be interchangeable or else the execution quits and previous values are kept.

You can set up a program that interchanges components through an INPUT statement structure or through a RELATION statement. When the feature of a parameter belongs to an assembly or to another component, an ADD COMPONENT statement or relation must include the component ID.

The format for assembly relation is:

```
XYZ = <parameter_name>:fid_<feature_name>:<comp_id>
```

OR

```
XYZ = <parameter_name>:fid_<N>:<comp_id>
```

where <comp_id> is the component ID in the assembly of the referenced part. To determine the component ID (<comp_id>) in Assembly mode, choose **Component Id** from the **RELATIONS** menu and use **Pick From File** to select the proper component.

To Interchange Components Programmatically

1. Include a string variable in the INPUT statement in an Assembly design.

   For example:

   ```
   INPUT
   fastener_name STRING
   "Enter name of fastener to be used in cam:"
   END INPUT
   ```

2. In the associated ADD statement, put the name of the string variable in parentheses.

For example, to add a part specifically to the assembly:

```
ADD PART (fastener_name)
...
END ADD
```

3. To interchange a part named washer for a subassembly or vice versa, use an ADD COMPONENT statement, using this format:

```
ADD COMPONENT (name with an extension, or variable)
COMPONENT ID <component Id>
```

For example:

```
ADD COMPONENT washer.prt
COMPONENT ID 4
...
END ADD
```

To Interchange Components Using Relations

1. In the INPUT statement, include a YES_NO variable.

For example:

```
INCL_CRANK YES_NO
```

2. Add an IF_ELSE clause in the RELATIONS.

For example:

```
RELATIONS
IF (INCL_CRANK == YES)
PART_NAME = "CRANK"
ELSE
PART_NAME = "SHAFT"
ENDIF
END RELATIONS
```

The ADD statement includes the variable defined in the ADD statement (it is enclosed in brackets).

For example:

```
ADD PART (PART_NAME)
...
END ADD
```

To Replace Family Table–Driven Components

You can automatically replace family table–driven components according to design criteria by using the `lookup_inst` function. With this function, you can search a component family table to find an instance that fits the values of the search parameters. If the lookup function does not find a match, it returns the name of the generic.

The format for `lookup_inst` is:

```
lookup_inst ("generic_name", match_mode, Òparam_name_1Ó, match_value_1,
Òparam_name_2Ó, match_value_2,...)
```

where

- `generic name`—Name of the generic model with a `.prt` or `.asm` extension
- `match_mode`—One of the following values:
  - –1 (find closest instance with `param` values less than or equal to supplied values)
  - 0 (find instance with `param` values that match supplied values exactly)
  - 1 (find closest instance with `param` values greater than or equal to supplied values)
- `param_name_1`—Family table parameter name
- `match_value_1`—Value to match against

Example: Replacing Family Table-Driven Components

Given an assembly that consists of a block and a peg, assemble the instance that matches the diameter of the hole in the block.

`inst_name` = declared string parameter initialized to generic part name

`generic_name` = `peg.prt`. This part contains a number of instances based on diameter dimension (d) and length dimension (d1).

Family instance names of `peg.prt` include:

```
2 x 4 - d0 = 2, d1 = 4
2 x 5 - d0 = 2, d1 = 5
2 x 6 - d0 = 2, d1 = 6
3 x 4 - d0 = 3, d1 = 4
3 x 5 ............
3 x 6 ............
```

Add a relation to the control in which `peg.prt` is added to an assembly controlled by dimensions of a feature in `block.prt`. The relation is:

```
inst_name = lookup_inst ("peg.prt", 0 , "d2", d6:0, "d1", d5:0 +1)
```

In this way, the instance of `peg.prt` being assembled to `blockpeg.asm` is controlled, based on the dimensions of the hole in `block.prt`.

The Pro/PROGRAM listing would look like this:

```
INPUT
END INPUT
RELATIONS
INST_NAME = LOOKUP_INST ("PEG.PRT", 0, "D2", D6:0, "D1", D5:0 + 1)
END RELATIONS
ADD PART BLOCK
INTERNAL COMPONENT ID 1
END ADD
ADD PART (INST_NAME)
INTERNAL COMPONENT ID 2
PARENTS = 1 (#1)
END ADD
MASSPROP
END MASSPROP
```

To Replace User-Defined Features

You can programmatically interchange user-defined features using a `CHOOSE` statement:

```
CHOOSE (<variable name>)
```

where `<variable name>` is the name of a string variable that contains the ID of the group to be placed. All the available IDs can be found in the `ADD` statement of the leader of the currently active group in the Pro/PROGRAM listing.

Example: Replacing User-Defined Features

```
INPUT
GROUP STRING
"ENTER GROUP TO PLACE 300/352/409"
END INPUT
CHOOSE (GROUP)
```

Note:

- To use a `CHOOSE` statement, you must first manually replace a family table instance of the group or replace the group with another group.
- `CHOOSE` statements cannot be included in conditional statements.

The group leader's `ADD` statement could look like this:

```
ADD FEATURE (initial number 4)
INTERNAL FEATURE ID 300
PARENTS = 33(#1)
TYPE = PROTRUSION
FORM = EXTRUDED
SECTION NAME = S2D0002
DEPTH = FROM SKETCH TO BLIND
FEATURE'S DIMENSIONS:
```

```
d44 (d23) = 2.00
d45 (d24) = 1.00
d46 (d25) = 1.00
d47 (d26) = 2.00
d48 (d27) = 1.00
MEMBER OF A GROUP, NAME = RECT
LEADING FEATURE OF THE GROUP: ID = 303
LAST FEATURE OF THE GROUP: ID = 303
GROUP IS REPLACEABLE BY FEATURES ID (NAME):
409(round) and 352 (circular)
END ADD
```

EXECUTE Statements

Using EXECUTE Statements in Assembly Listings

EXECUTE statements are valid for assembly listings only. They provide a link between input variables in an assembly and input variables for programs in parts and in the subassemblies that make up the assembly. EXECUTE statements follow this sequence:

```
EXECUTE {PART} name or variable
                  {ASSY }

input variable of design at next lower level = expression

input variable.....

END EXECUTE
```

Similar to an ADD statement, an EXECUTE statement can be used in the format EXECUTE COMPONENT to interchange parts and assemblies. When specifying the component, make sure to use its extension (.prt or .asm).

Note: When you are running a program, each part can be executed (that is, each part can receive variable values through an EXECUTE statement) only once. Avoid including conflicting instructions.

Hierarchy of Assembly Execution

Assemblies can execute subassemblies, which in turn can execute other subassemblies. The parts that compose a subassembly are not executed by the main assembly but are instead executed by the subassembly. Only the next level down in an assembly is executed by the assembly design.

Transferring Input Values from the Upper-level Assembly

The input variables are used to transfer input data from the upper-level assembly to the appropriate parts and subassemblies to drive the creation of the model.

For example, for the part `block_base`, the listing looks like this:

```
INPUT
key_size
ansi_thread
...
END INPUT
RELATIONS
d5 = key_size
d3 = depth * 1.25
END RELATIONS
....
```

Then the design listing for the assembly looks like this:

```
INPUT
hole_diameter NUMBER
thread_type STRING
depth
...
END INPUT
RELATIONS
END RELATIONS
EXECUTE PART block_base
key_size = hole_diameter/2 + 0.025
ansi_thread = thread_type
depth = DEPTH
...
END EXECUTE
```

And the design for the part `block_base` looks like this:

```
INPUT
ADD FEATURE.....
```

Note:

- The parameter `key_size` appears in the `EXECUTE` statement for the assembly and the `INPUT` statement for the part. This is necessary for the parameter value to be passed down from the assembly to the part. If the parameter does not appear in both places, or no `EXECUTE` statement is in the assembly design for the part, then those values that are currently in memory are used for the part.
- The parameter `thread_type` is set equal to `ansi_thread` in the `EXECUTE` statement, which is then passed to the part through the `INPUT` statement.
- The parameter `depth` is set equal to `DEPTH` in the `EXECUTE` statement and passed to the part using the same name in the `INPUT` statement. This technique is often preferable to step 2 because it is easier to keep track of the parameters.

- The relation `d5 = key_size` is not necessary. The parameter symbol `d5` can instead be renamed `key_size` using the **Symbol** command in the **DIM COSMETIC** menu.

Specifying a Part to Execute in an Assembly Program

When you are using an assembly program to replace a part in the assembly using interchangeability records, you can make sure that the appropriate part program is executed by entering the part name as a variable in the `EXECUTE` statement. This operation is similar to using a variable in an `ADD PART` statement.

For example, an assembly program could look like this:

```
INPUT
COMPONENT STRING
"Enter part name"
DIAMETER NUMBER
END INPUT
....
EXECUTE PART (COMPONENT)
d1=DIAMETER
END EXECUTE
```

If an `EXECUTE` statement passes values to variables `A` and `B`, and an `INPUT` statement declares only the variable `A`, the following occurs:

- A warning message informs you that the variable `B` has not been defined. You can then edit your design to correct the error.
- If you incorporate changes in the model after ignoring the warning, the value of `A` is passed to a parameter with the same name in the part being executed.

Using EXECUTE Statements inside IF-ENDIF Statements

`EXECUTE` statements can be used inside `IF-ENDIF` statements as a way to avoid execution of the lower-level model, unless necessary. If not executed, the current values of the model are used.

For example:

```
INPUT
key YES_NO
"Does the assembly have a key (Y/N):"
IF key == YES
key_name STRING
"Enter key name:"
ENDIF
END INPUT
RELATIONS
END RELATIONS
IF key == YES
EXECUTE PART (key_name)
```

```
END EXECUTE
ENDIF
```

The part keyname is executed only if it is included in the assembly.

Mass Properties and INTERACT Statements

Updating Mass Properties When Geometry Changes

Use the MASSPROP statement to update mass properties each time geometry changes. After you have specified parts or assemblies for which mass properties are to be updated, you can request the current value of a required parameter through the relations mechanism.

To update mass properties, use the following format:

```
MASSPROP
PART NAME
ASSEMBLY NAME
END MASSPROP
```

Note: When specifying the model for which mass properties are to be calculated, enter the model name without an extension.

The MASSPROP statement can contain the IF... ELSE clause. If you add a condition to the MASSPROP statement, the mass properties of an object will be calculated only if that condition is met.

For example:

```
MASSPROP
IF THICKNESS > 1
PART PLATE
ELSE
ASSEMBLY BLOCK
ENDIF
END MASSPROP
```

In the preceding example, if the parameter THICKNESS is more than 1, mass properties is recalculated for the part PLATE; otherwise, mass properties for the assembly BLOCK is calculated.

Using INTERACT Statements as Place Holders

INTERACT statements provide a placeholder for creating interactive part and assembly features. They can be inserted anywhere within the FEATURE ADD - END ADD or PART ADD - END ADD statement.

For example, the ELSE statement in the previous example could have been constructed as follows:

```
ADD PROTRUSION.....
IF d1 > d2
ADD HOLE.....
```

```
ELSE
INTERACT
END IF
ADD CUT.....
```

In this example, an alternate set of features is to be created if `d1` is not greater than `d2`.

Interact mode works similarly to Insert mode accessed from Pro/ENGINEER.

Executing an INTERACT Statement

When the system encounters an `INTERACT` statement in the program, the execution of the program is interrupted. At this point, you can add new features. Also at this point, the system displays an incomplete model built up according to the last instruction before the `INTERACT` statement. In Interact mode, the model is frozen and cannot be modified.

After you are in Interact mode, select any feature you want to add from the **FEAT CLASS** menus and proceed to specify all required parameters. After you have created a new feature, the system asks whether you want to continue adding features. If you answer `N`, program execution resumes. After execution is completed, any new features added within the `INTERACT` statement replace the `INTERACT` statement in the model design.

Note: You can quit interacting immediately after the program moves into the `INTERACT` phase (before you start to create features). Choose **Done/Return** from the **FEAT CLASS** menu and answer `N` to the system prompt asking if you want to continue. The program resumes execution and proceeds to the end.

Feature Operations

To Suppress Part or Assembly Features

1. To suppress a part or assembly feature or components, add the word `SUPPRESSED` immediately following the word `ADD`:

```
ADD SUPPRESSED PROTRUSION
```

2. To resume a suppressed feature, delete the word `SUPPRESSED` from the `ADD FEATURE` clause.

Note: Suppression through the use of Pro/PROGRAM works the same way as in regular Pro/ENGINEER (suppressed models are not retrieved when an assembly is retrieved). Therefore, suppressed models are not stored when you save an assembly with the **Save As** command.

To Suppress and Resume Individual Group Members

In Pro/PROGRAM, to suppress a single feature that is *not* part of a group, add the word `SUPPRESSED` after the word `ADD`, as shown in the following example. Then you would add a line to the end of the feature that reads `END ADD`.

```
ADD SUPPRESSED FEATURE
INTERNAL FEATURE ID  303
PARENTS = 240(#8)

ROUND: General

NO. ELEMENT NAME    INFO                                          STATUS
--- ------------ -----------                                      ------
 1  Round Type   Simple                                          Defined
 2  Attributes   Constant, Edge Chain                            Defined
 3  References                                                   Defined
 4  Radius       Value = 15.0000                                 Defined
 5  Round Extent                                                 Optional
 6  Attach Type  Make Solid - Feature has solid geometry         Defined


FEATURE'S DIMENSIONS:
d26 = 15R
END ADD
```

To suppress an entire group programmatically, include the statement `IF <value = NO for a yes/no parameter>` before the `GROUP HEADER` line, and the syntax `ENDIF` after the last line in the group. All members of the group are suppressed.

Note: In Pro/PROGRAM, all groups contain a group header, which is identical to the group name.

Suppressing Single Features That Are Members of a Group

You can suppress and resume individual features that are members of a group, provided you have set the configuration option `del_grp_memb_ind` to `yes`. To suppress group members in Pro/PROGRAM, use the same syntax that you use to suppress an entire group, as shown in the preceding paragraph. However, the placement of the syntax is different. Instead of placing the lines before and after the entire group, you place them before and after the single feature within the group, as though you were suppressing any other single feature that is not a group member.

To resume the individual feature, delete the lines `IF` <value = `no`> and `ENDIF` from the beginning and end of the feature.

Note: In order to suppress individual members of a group, you must set the configuration option `del_grp_memb_ind` to `yes`.

To Change Feature Dimensions

You can change the dimensions of features in the program by replacing a `DIMENSION` statement with:

```
MODIFY d# = value
```

You can also assign a new dimension value through the `RELATIONS` statement.

Editor Errors

Editing a Design to Correct an Error

Editor errors that prevent Pro/PROGRAM from reading the design are caught as soon as you exit the editor. Some ways that errors can occur are:

- Having an `IF` statement without an `END IF` statement or vice versa.
- Typing a variable name incorrectly in a relation or a condition.
- Reordering a child before the parent.
- Deleting a parent feature.

If the file contains errors, the **PROG ERROR** menu appears with the following active commands:

- **Abort**—Cancel changes that you have made to the design and keep it as it was prior to editing.
- **Edit**—Edit the design to correct errors. Error messages indicate the location and type of error. These messages are ignored during subsequent design processing; they are deleted if new errors are found and inserted into the design, or if you exit from the model.

Index