Windchill REST Services User's Guide

1.5

Copyright © 2019 PTC Inc. and/or Its Subsidiary Companies. All Rights Reserved.

User and training guides and related documentation from PTC Inc. and its subsidiary companies (collectively "PTC") are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed software user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. Training materials may not be copied without the express written consent of PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes. Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC.

UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.

PTC regards software piracy as the crime it is, and we view offenders accordingly. We do not tolerate the piracy of PTC software products, and we pursue (both civilly and criminally) those who do so using all legal means available, including public and private surveillance resources. As part of these efforts, PTC uses data monitoring and scouring technologies to obtain and transmit data on users of illegal copies of our software. This data collection is not performed on users of legally licensed software from PTC and its authorized distributors. If you are using an illegal copy of our software and do not consent to the collection and transmission of such data (including to the United States), cease using the illegal version, and contact PTC to obtain a legally licensed copy.

Important Copyright, Trademark, Patent, and Licensing Information: See the About Box, or copyright notice, of your PTC software.

UNITED STATES GOVERNMENT RIGHTS

PTC software products and software documentation are “commercial items” as that term is defined at 48 C.F. R. 2.101. Pursuant to Federal Acquisition Regulation (FAR) 12.212 (a)-(b) (Computer Software) (MAY 2014) for civilian agencies or the Defense Federal Acquisition Regulation Supplement (DFARS) at 227.7202-1(a) (Policy) and 227.7202-3 (a) (Rights in commercial computer software or commercial computer software documentation) (FEB 2014) for the Department of Defense, PTC software products and software documentation are provided to the U.S. Government under the PTC commercial license agreement. Use, duplication or disclosure by the U.S. Government is subject solely to the terms and conditions set forth in the applicable PTC software license agreement.

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

Contents

1

Windchill REST Services Overview

This section explains the basics of REST, OData, and Windchill REST Services.

REST

Representational state transfer or REST is an architectural pattern for web services. In this architecture, business objects on the server are represented as web resources. Client act on these web resources using the HTTP verbs such as, GET, POST, PATCH, PUT, and DELETE.

For example, consider a RESTful web service for parts that exposes a web resource `/Parts`. To get a list of parts, the clients of this web service send an HTTP GET request to `/Parts`. To create a part, the clients send a POST request to `/Parts` and specify a payload of the attribute values that are required to create the part.

OData

OData (Open Data Protocol) is an ISO/IEC approved, OASIS standard for building and consuming RESTful web services. OData enables exchange of data across web clients using HTTP messages.

The OData protocol mandates that a compliant web service must:

- declare an Entity Data Model (EDM) at a well-known URL.
- provide a uniform way to form URLs for entities and entity sets defined in the EDM.
- enable clients to send HTTP requests POST, GET, PATCH, PUT, and DELETE on entity and entity set URLs for creating, reading, updating, and deleting entities.
- support request headers and query parameters for client interaction as defined in the standard.

Please refer to the following resources for more information on OData, version 4.0:

- OData.org—Documentation on basics of OData.
- OData Protocol—Documentation of the protocol.
- Common Schema Definition Language (CSDL)—Documentation of the format used to document the EDM of a service.
- URL Conventions—Documentation of how to form URLs for entities and entity sets of an OData service.

Windchill REST Services

Windchill REST Services is a module that enables developers to configure OData services in Windchill. This module has its own release cycle, independent of the Windchill release. From Windchill 11.0 M030 CPS06 onward, the module is

bundled with Windchill CPS. The module comprises of a framework and a set of PTC domains. The domains are configured using the functionality available in the framework.

Framework

The framework is designed to read a set of configuration files. These configuration files have the information required to set up OData services on Windchill. An OData service in Windchill REST Services is called a domain. Domains expose Windchill object types and object collections as OData entity types and entity sets.

OData standard specifies the URLs to access entities and entity sets in a domain. The URLs provide a uniform interface between domains and clients.

The framework generates the Entity Data Model (EDM) for each domain when a request is made to the `$metadata` URL of a domain. The framework also generates the service document for each domain when requested by clients.

The framework processes the HTTP requests made by clients to entity collections and entity instances in a domain. For example, when a client makes a GET request to an entity set, the framework processes the request in the following way:

- checks the Windchill type for entities contained in the entity set,
- queries Windchill to obtain the objects of the type,
- converts the objects to OData entities,
- sends the entities back to the client as a response to the GET request.

Similarly, the framework processes the POST, PATCH, PUT, and DELETE requests.

The framework is designed to be extensible. It enables overrides and enhancements to the processing logic of requests. The request processing infrastructure of the framework has built-in code hooks. The domain authors can use the hooks to either override or enhance the default processing logic of the requests executed by the framework. The implementation of these hooks is done in JavaScript and is executed by the Nashorn engine available in Java 8. You can work with Windchill Java APIs while writing the JavaScript implementation of the hooks.

PTC Domains

Windchill REST Services includes a set of domain configurations for specific functional areas of Windchill. The schema of Windchill consists of a broad range of functional areas. The domains are designed to enable access to smaller and functionally independent areas of this schema to RESTful clients.

The following domains are available with Windchill REST Services:

- PTC Product Management Domain
- PTC Document Management Domain
- PTC Data Administration Domain
- PTC Principal Management Domain
- PTC Common Domain
- PTC Navigation Criteria Domain
- PTC Dynamic Document Management Domain
- PTC Parts List Management Domain
- PTC Service Information Management Domain
- PTC Quality Domains
    - PTC Quality Management System Domain
    - PTC NC (Nonconformance) Domain
    - PTC CAPA Domain
    - PTC Customer Experience Management (CEM) Domain
    - PTC Regulatory Master Domain
    - PTC Audit Domain
- PTC Info*Engine System Domain
- PTC Factory Domain
- PTC Manufacturing Process Management Domain
- PTC Change Management Domain
- PTC Classification Structure Domain
- PTC Saved Search Domain
- PTC Visualization Domain
- PTC Product Platform Management Domain
- PTC CAD Document Management Domain
- PTC Event Management Domain
- PTC Supplier Management Domain
- PTC Workflow Domain

These domains are extensible by design. You can add new entities and expand existing entities in the domains. Refer to the section PTC Domains on page 74 for more information about domains.

For more information on how to extend entities in PTC-provided domains, please see the section Extending Domains on page 206.

Custom Domains

Windchill REST Services enables you to create new domains. You can create new domains in any functional areas, even the areas which are not available in the PTC domains. Refer to the section Customizing Domains on page 206, for more information on how to create custom domains.

Code Sample

Windchill REST Services User's Guide includes examples and code snippets for better understanding of the concepts. The examples in the section *Examples for Performing Basic REST Operations*, demonstrate the basic operations in Windchill REST Services.

However, please note that 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.

2

Installing Windchill REST Services

Installation Prerequisites

Windchill REST Services has its own independent release cycle. It is bundled with Windchill CPS releases.

The Windchill software matrix lists the Windchill release and the version of the bundled Windchill REST Services module. The latest software matrix is available at the following URL:

http://www.ptc.com/appserver/cs/doc/refdoc.jsp

This URL directs you to the PTC Online Support Web page of reference documents. For the software matrix document, specify the following search criteria:

1. In the **Product** list, select **Windchill REST Services**.
2. In the **Release** list, select the current release number.
3. Select **Matrices** from the **Document Type** list.

The Windchill software matrix opens.

Installation Process

Windchill REST Services is installed and updated with the PTC Solution Installer (PSI). Windchill REST Services is installed with a full or CPS install of Windchill.

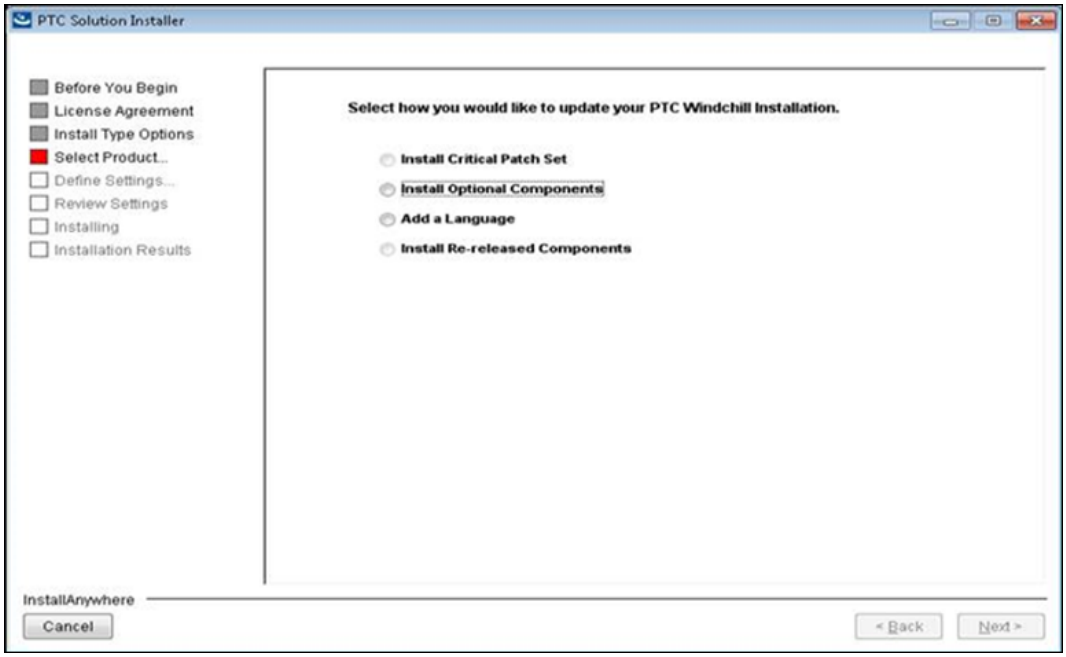
From Windchill 11.1 F000 onward, Windchill REST Services is a mandatory component of PTC Solution Installer (PSI) and is installed with a new installation. It must be available in the staging area while installing Windchill. The relevant releases of Windchill REST Services are bundled with Windchill 11.1 F000 CPS CDs and are updated when a CPS is installed.

For Windchill 11.0 M030 release, from F000 to CPS05, Windchill REST Services was not available. From Windchill 11.0 M030 CPS06 onward, Windchill REST Services is bundled with the CPS. When you install Windchill 11.0 M030 CPS06, Windchill REST Services is also installed.

For Windchill 11.0 M020 release, Windchill REST Services is not available for installation with the PSI and must be installed as a standalone with a command line utility. Refer to the article available on Technical Support at:

https://support.ptc.com/search/Pages/results.aspx?k

.



When you update a Windchill installation with a CPS release, select the **Install Critical Patch Set** option to install or update Windchill REST Services.

3

Windchill REST Services Framework Capabilities

Overview

Windchill REST Services has a framework that provides capabilities to create OData services based on the OData V4 protocol.

The framework enables domain authors to create a set of configuration files that define the OData entities available in a service. You can map the OData entities to Windchill types.

The framework provides default processing logic for GET, POST, PUT, PATCH, and DELETE requests. The domain authors can override or enhance the processing logic with code hooks.

The framework supports inheritance. Domain entities can derive properties, actions, or functions from the framework without defining them explicitly. For example, consider a Windchill capability `Workable`. Domain entities that inherit the framework capability `Workable` also inherit the properties of the capability. The properties to display version, iteration, actions to check out, check in, undo check out, and revise the entity are inherited.

Support for OData

The framework is designed to support OData Protocol V4. All aspects of OData standard are not currently supported by Windchill REST Services. PTC intends to support minimum OData V4 compliance in subsequent releases of Windchill REST Services.

The key features of OData that are supported in the framework are discussed in the subsequent sections.

OData Services as Domains

The framework enables you to set up domains which contain entities that are mapped to Windchill types. Domains are equivalent to OData service root.

Similar to the OData web service, a domain can be accessed at the Domain Root URL, which is of the form `https://<Windchill server>/<Windchill App Context>/servlet/odata/<Domain Version>/<Domain Identifier>/`

where,

- `<Windchill App Context>` is Windchill in a standard installation
- `<Domain Version>` is the version of the domain API and can be `v1`, `v2`, and so on. `<Domain Version>` is optional in the URL and must be specified only if the client needs a specific version of the domain.
    - If `<Domain Version>` is not specified in the URL, the framework checks if the domain version is specified in the `Accept` header.

    - If <Domain Version> is not specified in the URL or in the Accept header, then the default configured version is used by the framework.
- <Domain Identifier> is the identifier for the domain. For example, the identifier for Product Management domain is ProdMgmt.

The URL is called at the Domain Root, which is equivalent to OData service root URL. A GET request to this URL returns the list of entity sets that are available in a domain.

For example:

- The Domain Root URL of the Product Management domain for version 1 is:

  ```
  https://windchill.ptc.com/Windchill/servlet/odata/v1/ProdMgmt/
  ```

- The Domain Root URL of the Product Management domain for the default version is:

  ```
  https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/
  ```

Entity Data Model of a Domain

An Entity Data Model (EDM) is the specification of entities that are available for a domain. Each entity is further defined by its structural and navigation properties. Structural properties have values. Navigation properties are references to other entities in the domain.

The EDM of a domain is defined in Common Schema Definition Language (CSDL). CSDL defines the entity model as an XML representation. The EDM of a domain can be accessed by adding $metadata at the end of the Domain Root URL. For example, the URL for EDM of the Product Management domain is:

```
https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/$metadata
```

Clients can send an HTTP GET request to this URL to get the EDM of the Product Management domain. The EDM enables clients to get more information about the entities, relationships, functions, and actions provided by the domain.

OData Primitives

OData primitives are the data types supported by the OData standard.

The following table lists the OData primitives that are supported by the framework.

The table also shows the recommended mapping of OData primitives to Windchill and Java types.

Framework Type	OData Type	Windchill/Java Type
SByte	Edm.SByte	byte, java.lang.Byte
String	Edm.String	char, java.lang.Character

Framework Type	OData Type	Windchill/Java Type
Int16	Edm.Int16	short, java.lang.Short
Int32	Edm.Int32	int, java.lang.Integer
Int64	Edm.Int64	long, java.lang.Long
Single	Edm.Single	float, java.lang.Float
Double	Edm.Double	double, java.lang.Double
Boolean	Edm.Boolean	Boolean, java.lang.Boolean
String	Edm.String	String
DateTimeOffset	Edm.DateTimeOffset	Timestamp

OData Query Parameters

Windchill REST Services supports the following query parameters from the OData standard:

- `$filter`—Query criteria to filter the entities in a collection. When you access an entity set, an expression can be specified in the `$filter` query parameter. With the expression, you can restrict an entity set to only show entities for which the expression evaluation results to true value. OData supports filter expressions for a broad range of primitives. The framework only supports the following subset of OData expressions:
  - Expressions that use String, Int16, Int32, Int64, Boolean, DateTimeOffset, Single and Double types
  - Expressions that use comparison operators EQ, NE, GT, LT, GE, LE
  - Expressions that use logical operators AND, OR
  - Expressions that use unary operator NOT
  - Expressions that use the methods `startswith`, `endswith` and `contains`

  Windchill REST Services supports `$filter` query parameter on navigation properties. See the sections Configuring Navigation Properties on page 43 and Support for $filter on Navigation Properties on page 21 for more details.

  OData enables services to specify the properties of an entity that must not be used in `$filter` expressions. OData services usually choose to restrict the use of certain properties in `$filter` expressions because of performance problems. The properties which are restricted for use in the `$filter` expressions are automatically specified by the framework for Windchill REST Services domains. The `FilterRestrictions` annotation defined in the Capabilities Vocabulary of OData is used to specify the restricted properties.

This annotation is applied to the entity sets available in the domain and is seen in the EDM. Properties that are specified in this annotation are the nonpersistent or server calculated properties of Windchill.

The framework prevents properties specified in this annotation to be queried in the `$filter` expressions. If clients build query expression with these properties, the server returns an error message. For example, the sample below shows `FilterRestrictions` annotation applied to the Parts entity set:

```
<EntitySet Name="Parts" EntityType="PTC.ProdMgmt.Part">
    …
    <Annotation Term="Org.OData.Capabilities.V1.FilterRestrictions">
      <Record>
        <PropertyValue Property="NonFilterableProperties">
            <Collection>
                <String>VersionID</String>
                <String>ChangeStatus</String>
                <String>CabinetName</String>
            </Collection>
        </PropertyValue>
      </Record>
    </Annotation>
    …
</EntitySet>
```

- `$select`—Comma-separated list of entity properties that must be returned as a part of the response. For example, in the URL you can list the Document attributes such as, `Name` and `CheckoutState`, to display only the name of the document and its checkout status in the response.
- `$top`—Returns the specified number (N) of entities from the top, that is, the first N entities, in a collection.
- `$skip`—Skips the specified number (N) of entities from the top of a collection, and displays the set of entities, N+1 entity onward.
- `$orderby`—Sorts the entities in ascending and descending order. The query parameter is supported for the following:
  - Primitive attributes—String, Boolean, Integer, Real Number (Double), Date and Time types. See the following sample URL: `https://windchill.ptc.com/Windchill/servet/odata/ProdMgmt/Parts?$orderby=<Primitive Property> <asc/desc>`
  - Complex types—The following properties of a complex type support sorting:
    - `QuantityOfMeasureType`—Supported for `value` property. The properties `Unit`, `Precision`, and `Display` do not support sorting.

- `EnumType`—Supported for `InternalName` property. The property `Display` does not support sorting.
- `Hyperlink`—Supported for `URL` property. The property `Label` does not support sorting.

See the following sample URL:

```
https://windchill.ptc.com/Windchill/servet/odata/ProdMgmt/Parts?$
orderby=<Complex Property>/<Property> <asc/desc>
```

Windchill REST Services supports `$orderby` query parameter on navigation properties. See the sections Configuring Navigation Properties on page 43 and Support for $orderby on Navigation Properties on page 24 for more details.

OData enables services to specify the properties of an entity that must not be used in `$orderby` expressions. The properties which are restricted for use in the `$orderby` expressions are automatically specified by the framework for Windchill REST Services domains. The `SortRestrictions` annotation is used to specify the restricted properties. This annotation is applied to the entity sets and complex types available in the domain, and is seen in the EDM. `$orderby` is not supported for the following properties. These properties are specified by the framework in the `SortRestrictions` annotation:

- Nonpersistent or server calculated properties of Windchill
- Properties of complex types that do not support sorting

The framework prevents properties specified in this annotation from being queried in the `$orderby` expressions. If clients build query expression with these properties, the server returns an error message.

For example, the sample below shows `SortRestrictions` annotation applied to the `QuantityOfMeasureType` complex type:

```
<ComplexType Name="QuantityOfMeasureType">
                <Property Name="Value" Type="Edm.Double"/>
                <Property Name="Unit" Type="Edm.String">
                    <Annotation Term="PTC.NonSortable"/>
                </Property>
                <Property Name="Precision" Type="Edm.Int64">
                    <Annotation Term="PTC.NonSortable"/>
                </Property>
                <Property Name="Display" Type="Edm.String">
                    <Annotation Term="PTC.ReadOnly"/>
                    <Annotation Term="PTC.NonSortable"/>
                </Property>
                <Annotation Term="Core.Description">
                    <String>Used to for Windchill Quantity Of Measure types.</String>
                </Annotation>
```

```
                <Annotation Term="Org.OData.Capabilities.V1.SortRestrictions">
                    <Record>
                        <PropertyValue Property="NonSortableProperties">
                            <Collection>
                                <String>Precision</String>
                                <String>Unit</String>
                                <String>Display</String>
                            </Collection>
                        </PropertyValue>
                    </Record>
                </Annotation>
            </ComplexType>
```

- `$count`—Returns the number of items in a collection. When you specify the `$count` parameter in the URL, it is mandatory to specify the value as either `true` or `false`. If no value is specified, an error message is returned.

  The count is returned in the response body as `"@odata.count": <count_value>`.

  The following URLs support `$count`:

  - URLs for direct entities
  - URLs for navigation properties
  - URLs for expanding navigation properties

  For example, in the following URL `$count` is specified as `true`. The URL returns the count of all the parts available on the server.
  `ProdMgmt/Parts?$count=true`

The query parameters can be used in the following situations:

- When you access the entity sets defined in the domain
- When you navigate to a collection of entities from a given entity
- When you expand navigation properties of entities

Examples of how to use query parameters:

- GET request:

  `ProdMgmt/Parts?$filter=startswith(Number, '00')`
  The response shows only those parts that start with '`00`' in the entity set.
- GET request:
  `ProdMgmt/Parts('wt.part.WTPart:99999')/Uses?$filter=Quantity eq 1&$top=1`

  The response shows the first `PartUse` entity from the collection of `PartUse` entities that have Quantity equal to one.
- GET request:
  `ProdMgmt/Parts?$expand=Uses($select=Quantity,TraceCode;$filter= Quantity gt 1)`

Send this request to server, if you want to expand `Uses` navigation to only show `Quantity` and `TraceCode` properties of `PartUse` entities that have quantity greater than one.

- GET request:
  `ProdMgmt/Parts('wt.part.WTPart:99999')/Uses?$orderby=Quantity`

  The response sorts the `PartUse` entities in ascending order of quantity.
- GET request:
  `ProdMgmt/Parts('wt.part.WTPart:99999')/Uses?$orderby=CreatedOn,Quantity`

  The response sorts the `PartUse` entities in ascending order of date on which the entities were created and then on quantity. This sorting is possible when two entities have the same `CreatedOn` date and different `Quantity`. The entities are sorted, based on the quantity. However, if the entities have different `CreatedOn` dates, then the entities are sorted based only on date property.
- GET request:
  `DocMgmt/Documents?$filter=startswith(Name,'Demo')&$orderby=ZipCodeIn asc&$top=2`

  The response shows the top two documents whose name starts with `Demo`. The documents are sorted in ascending order of `ZipCodeIn`.
- GET request:
  `ProdMgmt/Parts('OR:wt.part.WTPart:897988')/UsedBy?$count=true`

  The example is created for `UsedBy` navigation. The navigation is available on the Part entity. The response returns the count as `"@odata.count": 2`.

```
{
  "@odata.context": "https://windchill.ptc.com/Windchill/servlet/odata/
                                          ProdMgmt/$metadata#Parts",
  "@odata.count": 2,
  "value": [
              {
               "@odata.type": "#PTC.ProdMgmt.Part",
               "ID": "OR:wt.part.WTPart:897456",
               ...
              },
              {
               "@odata.type": "#PTC.ProdMgmt.Part",
               "ID": "OR:wt.part.WTPart:897234",
               ...
              }
    ]
}
```

- GET request:
  `ProdMgmt/Parts?$expand=UsedBy($count=true)`

The example is created for expanding the `UsedBy` navigation. The navigation is available on the Part entity. The response returns the count as `UsedBy@odata.count": 2`.

```
{
 "@odata.context": "https://windchill.ptc.com/Windchill/servlet/odata/
                                    ProdMgmt/$metadata#Parts",
  "value": [
   {
     "@odata.type": "#PTC.ProdMgmt.Part",
     "ID": "OR:wt.part.WTPart:89798",
     ...
     "UsedBy@odata.count": 2,
     "UsedBy": [
   {
     "@odata.type": "#PTC.ProdMgmt.Part",
     "ID": "OR:wt.part.WTPart:897456",
     ...
   },
  {
     "@odata.type": "#PTC.ProdMgmt.Part",
     "ID": "OR:wt.part.WTPart:897234",
     ...
           }
        ]
     }
    ]
}
```

Support for $filter on Navigation Properties

The Windchill REST Services framework supports `$filter` query parameter for some navigation properties in the PTC Document Management, PTC Product Management, and PTC Change Management domains. The `$filter` options such as `contains`, `startswith`, and so on, are supported. This section describes the navigation properties that support the `$filter` query parameter.

See the section Configuring Navigation Properties on page 43 for more information on setting up navigation properties.

Context

You can query for a collection of change objects, parts, or documents using the `$filter` parameter on the `Context` navigation property. The filtering is supported for the `Name` property of the `Container` entity.

For example:

- `/DocMgmt/Documents?$filter=contains(Context/Name,'<substring_of_containername>')`

  This URL returns all the documents that are available in all the containers with a name that contains the specified substring. For example, if you specify the substring as `Cart`, then the URL returns all the documents that are available in all the containers with a name that contains the substring `Cart`.

- `/ProdMgmt/Parts?$filter=Context/Name eq '<container_name>'`

  This URL returns all the parts that are available in the specified container.

- `/ChangeMgmt/ChangeRequests('<change_request_ID>')?$expand=AffectedObjects($filter=Context/Name eq 'Power System')`

  This URL returns all the affected objects that have the context name set as `Power System` in the specified change request ID.

Organization

You can query for a collection of change objects, parts, or documents using the `$filter` parameter on the `Organization` navigation property. The filtering is supported for the `Name` property of the `Organization` entity.

For example:

`/DocMgmt/Documents?$filter=Organization/Name eq '<organization_principal_name>'`

This URL returns all the documents that are available in the specified organization.

Folder

You can query for a collection of parts or documents using the `$filter` parameter on the `Folder` navigation property. The filtering is supported for `Name`, `Location`, and `Description` properties of the `Folder` entity.

For example:

`/ProdMgmt/Parts?$filter=Folder/Description eq '<folderdescription>'`

This URL returns parts that are available in the folders with the specified folder description.

Attachments

You can query for a collection of change objects or documents using the `$filter` parameter on the `Attachments` navigation property.

The filtering is supported for `FileName`, `Description`, `Format`, and `FileSize` properties of the associated attachment.

For example:

- `/DocMgmt/Documents?$filter=Attachments/any(d:d/PTC.ApplicationData/Description eq '<attachmentdescription>')`

  This URL returns all the documents that contain the specified description for attachments.

- `/DocMgmt/Documents?$filter=Attachments/any(d:contains(d/PTC.ApplicationData/Format,'<attachmentformat>'))`

  This URL returns all the documents that have attachments in the specified type of format.

- `/ChangeMgmt/ChangeRequests('<change_request_ID>')/Attachments/PTC.ApplicationData?$filter=contains(FileName,'xyz.pdf')`

  This URL returns all the attachments that contain the specified file name for the specified change request.

PrimaryContent

You can query for a collection of documents using the `$filter` parameter on the `PrimaryContent` navigation property.

The filtering is supported for `FileName`, `Description`, `Format`, and `FileSize` properties of the associated primary content.

For example:

`/DocMgmt/Documents?$filter=PrimaryContent/PTC.ApplicationData/FileSize eq <doublevalue>`

This URL returns all the documents with primary content of the specified size.

Representation

You can query for a collection of parts using the `$filter` parameter on the `Representation` navigation property.

The filtering is supported for `Name`, `Description`, `DefaultRepresentation`, `FormatName`, and `OutOfDate` properties of the associated representation.

For example:

- `/ProdMgmt/Parts?$filter=Representations/any(d:d/OutOfDate eq true)`

This URL returns all the parts with representations that are out-of-date.

- `/ProdMgmt/Parts?$filter=Representations/any(d:startswith(d/Name,'<substring_of_representation_description>'))`

  This URL returns all the parts that have representations with the specified representation description.

Support for $orderby on Navigation Properties

The Windchill REST Services framework supports `$orderby` query parameter for following navigation properties in the PTC Document Management, PTC Product Management, and PTC Change Management domains. This section describes the navigation properties that support the `$orderby` query parameter.

Context

You can sort a collection of change objects, documents, or parts using the `$orderby` parameter on the `Context` navigation property. The sorting is supported for the `Name` property of the `Container` entity.

For example:

- `/DocMgmt/Documents?$orderby=Context/Name asc`

  This URL performs an ascending sort on documents, based on the name of the context.

- `/ProdMgmt/Parts?$orderby=Context/Name desc`

  This URL performs a descending sort on parts, based on the name of the context.

- `/ChangeMgmt/ChangeRequests('<change_request_ID>')?$expand=AffectedObjects($orderby=Context/Name desc)`

  This URL performs a descending sort on all the affected objects on the context name.

Organization

You can sort a collection of change objects, documents, or parts using the `$orderby` parameter on the `Organization` navigation property. The sorting is supported for the `Name` property of the `Organization` entity.

For example:

`/DocMgmt/Documents?$orderby=Organization/Name desc`

This URL performs a descending sort on documents, based on the name of the organization.

Folder

You can sort a collection of parts or documents using the `$orderby` parameter on the `Folder` navigation property. The sorting is supported for `Name`, `Location`, and `Description` properties of the `Folder` entity.

For example:

```
/ProdMgmt/Parts?$orderby =Folder/Location asc
```

This URL performs an ascending sort on parts, based on the location of the folder.

Attachments

You can sort a collection of change objects or documents using the `$orderby` parameter on the `Attachments` navigation property.

The sorting is supported for `FileName`, `Description`, `Format`, and `FileSize` properties of the associated attachment.

For example:

- `/DocMgmt/Documents('<document_ID>')?$expand=Attachments/PTC.ApplicationData($orderby=FileName asc)`

  This URL performs an ascending sort on all the attachments by their file names.

- `/ChangeMgmt/ChangeRequests('<change_request_ID>')?$expand=Attachments/PTC.ApplicationData($orderby=FileSize desc)`

  This URL performs a descending sort on all the attachments by their file size.

PrimaryContent

You can sort a collection of documents using the `$orderby` parameter on the `PrimaryContent` navigation property.

The sorting is supported for `FileName`, `Description`, `Format`, and `FileSize` properties of the associated primary content.

Representation

You can sort a collection of parts using the `$orderby` parameter on the `Representation` navigation property.

The sorting is supported for `Name`, `DefaultRepresentation`, `FormatName`, and `OutOfDate` properties of the associated representation.

For example:

```
/ProdMgmt/Parts('<part_ID>')?$expand=Representations($orderby=Name asc)
```

This URL performs an ascending sort on all the representations of the part by their name.

Querying with the DateTimeOffset Property

Windchill REST Services enables `DateTimeOffset` property to be used in `$filter` expressions. Windchill enables `DateTimeOffset` properties to be persisted in the database. However, it only allows the date portion of the `DateTimeOffset` property to be used in queries. Windchill stores all the `DateTimeOffset` properties in UTC format.

When you query with `DateTimeOffset` property in `$filter` expressions, you must specify both the date and time in UTC format. Windchill REST Services returns an error if the date and time properties in `$filter` expressions are specified in any other time zone. The `$filter` expressions with a `DateTimeOffset` property are evaluated only on the date portion of the property. For example, if you specify the expression `$filter=CreatedOn eq 2018-01-20T01:52:16Z`, the service qualifies all the entities created on January 20, 2018, in the UTC time zone regardless of the time specified in the expression.

For example, the URL used in `$filter` expression with `DateTimeOffset` property is:

```
https://windchill.ptc.com/Windchill/servet/odata/ProdMgmt/Parts?$filter=CreatedOn eq 2018-01-20T00:00:00Z
```

OData Prefer Headers

OData enables clients to request a preferred service behavior by specifying `Prefer` header in requests. In some cases, the service may not apply the preference specified by the client. However, the service in its response indicates the preference that was applied. OData supports many `Prefer` header values.

This section provides information about the preferences supported by Windchill REST Services.

Preference maxpagesize

The `odata.maxpagesize` preference enables clients to change the pagination behavior of the server. By default, services built with Windchill REST Services have a page size of 25 entities. If the client does not specify this preference, then in response to a GET request for a collection, Windchill REST Services generates a nextLink URL in the response if the number entities in the collection are greater than 25. Clients can use this URL to get to the next page of entities in the collection. With the `odata.maxpagesize` preference, the client can change the page size of the server to a maximum value of 200 entities.

For example, consider a client that specifies the following header while accessing a collection of 300 entities:

```
Prefer: odata.maxpagesize=100
```

In this case, the server changes its default page size and generates the nextLink after 100 entities in a collection. In its response, the server sends back the prefer header `odata.maxpagesize=100`, which indicates that the preference is applied. Use the nextLink to get the next 100 entities.

In the same example, if the client specifies `odata.maxpagesize=250`, the server restricts the page size to 200 entities in the collection. It generates a nextLink URL to the next 100 and sends back the `odata.maxpagesize=200` indicating the preference that was applied.

Preferences return=representation and return=minimal

The `return=representation` and `return=minimal` preferences enable clients to decide if the response should contain the modified content. If none of these preferences are specified, then the default preference `return=representation` is used to return the content.

- `return=representation`—The request returns the modified content in the body of the response. It also returns the appropriate HTTP status code.
- `return=minimal`—The request does not return the modified content in the body of the response. It only returns the appropriate HTTP status code.

PTC Annotations

The OData protocol supports specifying elements in EDM with custom annotations that provide additional information to the clients. The following annotations are available in Windchill REST Services:

Core.Description

This annotation is available on all entity types. It describes the entity.

```
<Annotation Term="Core.Description">
    <String>Some description</String>
</Annotation>
```

PTC.Operations

This annotation specifies the list of supported CRUD operations for entities. In the example below, the annotation is applied to the `Part` entity in the PTC Product Management domain:

```
<EntityType Name=”Part>
<Key>
    <PropertyRef Name=”ID”/>
</Key>
<Property Name=”ID” Type=”EDM.String”>
…
  <Annotation Term="PTC.Operations">
   <String>READ, CREATE, UPDATE, DELETE</String>
  </Annotation>
</EntityType>
```

The annotation indicates that the framework supports reading, creating, updating, and deleting parts.

PTC.MultiOperations

This annotation is available on an entity type. It specifies the list of CRUD operations for which actions are available for multiple objects. For example, if CREATE operation is listed in the annotation for a `Part` entity type, then a bound action `CreateParts` is available for the `Part` entity.

```
<Annotation Term="PTC.MultiOperations">
    <String>CREATE,UPDATE,DELETE</String>
</Annotation>
```

PTC.ReadOnly

This annotation indicates entity properties that are read-only. In the example below, an annotation from the EDM of the PTC Product Management domain is shown:

```
<EntityType Name=”Part”>
<Key>
    <PropertyRef Name=”ID”/>
</Key>
<Property Name=”ID” Type=”EDM.String”>
    …
<Property Name=”State” Type=”PTC.EnumType”>
        <Annotation Term="PTC.ReadOnly"/>
</Property>
…
```

```
</EntityType>
```

The annotation indicates that the property `State` on the `Part` entity is read-only.

PTC.SecurityLabel

This annotation indicates security labels. In the example below, an annotation from the EDM of the PTC Product Management domain is shown:

```
<EntityType Name=”Part”>
    <Property Name="<Internal_name_for_Security_label>" Type="Edm.String">
      <Annotation Term="PTC.ReadOnly"/>
      <Annotation Term="PTC.SecurityLabel"/>
    </Property> …
…
</EntityType>
```

PTC.UpdateableViaAction

This annotation indicates that you cannot update the property using a PATCH request. You can modify the property using the `UpdateCommonProperties` action. In the example below, an annotation from the EDM of the PTC Product Management domain is shown:

```
<EntityType Name=”Part”>
 <Property Name="EndItem" Type="Edm.Boolean" Nullable="false">
  <Annotation Term="PTC.UpdateableViaAction">
   <String>Property is updateable via UpdateCommonProperties action</String>
  </Annotation>
 </Property>
<NavigationProperty Name="Organization" Type="PTC.PrincipalMgmt.Organization">
  <Annotation Term="PTC.UpdateableViaAction">
   <String>Property is updateable via UpdateCommonProperties action</String>
  </Annotation>
</NavigationProperty>
…
</EntityType>
```

Org.OData.Capabilities.V1.SortRestrictions

This annotation is available on an entity set. It specifies the properties of an entity set that you cannot use for sorting the entity set.

```
<Annotation Term="Org.OData.Capabilities.V1.SortRestrictions">
  <Record>
     <PropertyValue Property="NonSortableProperties">
       <Collection>
         <String>Property Name 1</String>
         <String>Property Name 2</String>
          …
       </Collection>
     </ PropertyValue>
  </Record>
</Annotation>
```

PTC.NonSortable

This annotation is deprecated. Use the annotation `Org.OData.Capabilities.V1.SortRestrictions` instead. The properties marked as nonsortable are shown in the `Org.OData.Capabilities.V1.SortRestrictions` annotation.

PTC.Capability

This annotation is available on an entity type. It designates an entity type that implements a specific capability. In the following example, the annotation shows the `Notifiable` capability, which indicates that you can subscribe to the entity type and the instances of the entity type using webhooks.

```
<Annotation Term="PTC.Capability">
    <String>Notifiable</String>
</Annotation>
```

PTC.ClassificationNameSpace

This annotation is available on a property of type `PTC.ClassificationInfo`. Clients can use the value from the annotation to specify the `clfStructureNameSpace` property while classifying a part.

```
<Annotation Term="PTC.ClassificationNameSpace">
    <String>com.ptc.csm.default_clf_namespace</String>
</Annotation>
```

API Catalog for Windchill REST Services Endpoints

The Windchill REST Services release includes an API catalog, which is a developer document. The catalog is a web page that is accessible from the Windchill user interface. It lists all the endpoints along with the supported operations. The catalog is the Swagger specification of the endpoints that are available in Windchill REST Services. You can interactively execute the HTTP operations on the endpoint URLs.

You can access the catalog from a running instance of Windchill. To access the catalog in Windchill perform the following steps:

1. To display the **Customization** icon on the **Browse** tab, in the **Preference Management** utility, set **Client Customization** to **Yes**.
2. In the **Browse** tab, click the **Customization** icon. The **Customization** page opens.
3. Click **Documentation**.
4. Under the **API** menu, click **OData REST APIs**. The Windchill REST Services catalog opens.

The catalog lists all the domains that are supported by your current installation of Windchill. The list of domains that appear in the catalog are generated based on a set of catalog configuration files. The configuration files are available in the PTC and custom configuration paths. There is one catalog configuration file for every domain. The name of the catalog configuration file is specified in the following format:

`config.<domain_name>.json`

where, `<domain_name>` is the name of the domain. For example, `ProdMgmt`, `DocMgmt`, `PrincipalMgmt`, and so on.

The JSON object in the configuration file contains key-value pairs that correspond to the following configurable options:

- `includeInDocumentation`—Includes the domain in the catalog. Specify as `true` to include the domain. It takes a Boolean value as input.

  If the configuration file is not defined for a domain, it does not appear in the catalog.
- `blackListedEndpoints`—Excludes endpoints from the catalog. It is a JSON array. Its value is set as an array of objects that contain `path` and `ops` keys.
  - `path` key is a regular expression of the endpoints that should be excluded.
  - `ops` key is an array of strings, which specifies the operations that should be excluded for the endpoints specified in `path`.

  Note

  The strings in the `ops` array are case-sensitive. For example, specify the operations as `["GET", "POST", "PATCH"]`. Do not specify the operations as `["get", "Post", "patch"]`.

- `navigationLevel`—Specifies the levels of navigation the catalog generator should follow to find endpoints for a given domain. It is an integer and is configurable up to 3 levels.
- `htmlInfoFileName`—Specifies the name of the static HTML file that is set as the summary page for the domain. The file opens when you click the name of the domain in the sidebar of the catalog and have not selected a version. The value that is set for this option corresponds to an HTML file that is available at:

  `<Windchill>/codebase/netmarkets/html/wrs/catalog`

  where, `<Windchill>` is the Windchill installation directory.

If this option is not specified or the HTML file specified in the option does not exist, the content area remains blank until a version is selected from the sidebar.

For example, a catalog configuration file for PTC Product Management domain can be defined as:

```
{
    "includeInDocumentation" : true,
    "blackListedEndpoints" :[
        {
          "path":"\\/.*/SmallThumbnails.*",
          "ops":["GET", "DELETE", "PATCH", "POST"]
        },{
          "path":"\\/.*/Thumbnails.*",
          "ops":["GET", "DELETE", "PATCH", "POST"]
        }
    ],
    "navigationLevel" : 1,
    "htmlInfoFileName" : "prodmgmtdomain.html"
}
```

The domains listed in catalog are extensible. You can include any custom domains configured in your installation in this list. To include a custom domain in the catalog, create a domain configuration file and add it at the following location:

`<Windchill>/codebase/rest/custom/doc`

The domain configuration files defined by PTC for the catalog are available at the following location:

`<Windchill>/codebase/rest/ptc/doc`

In addition to the endpoints that are automatically generated, you can extend the catalog with manually authored Swagger specification. This enables you to add additional endpoints to your custom domains. You can also override the existing endpoints in the PTC provided domains.

Every version of each domain can have its own file for additional endpoints. You should define these endpoints in a separate JSON file. The naming convention for these JSON files is:

`doc.<Version>.<domain_name>.json`

For example, if you added or updated endpoints in the PTC Product Management domain for `v1` version, the name of the file should be `doc.v1.ProdMgmt.json`.

Note

Save the Swagger specification file in JSON format.

All user-specified Swagger specification files that extend PTC or custom domains, should be at:

`<Windchill>/codebase/rest/custom/doc`

See the Swagger documentation for more information about Swagger Specification and Swagger Editor.

In the catalog, the PTC provided endpoints that are automatically generated or manually authored by PTC in the Swagger specification files, are listed under **Service Endpoints**. The endpoints that are created by users at `<Windchill>/codebase/rest/custom/doc` are listed under **Custom Endpoints** in the catalog.

Note

All possible combinations of OData URLs are not available. Only a subset of URLs is available in the catalog.

Domain Configuration

This section describes how to configure a domain in Windchill REST Services. The folder structure and files that are required to configure a domain are explained in detail.

Configuration Paths and Files

The configuration home of Windchill REST Services is `<Windchill>/codebase/rest` folder. Windchill REST Services searches for the configuration files in a set of subfolders which is called the configuration path set. These subfolders are available in the configuration home.

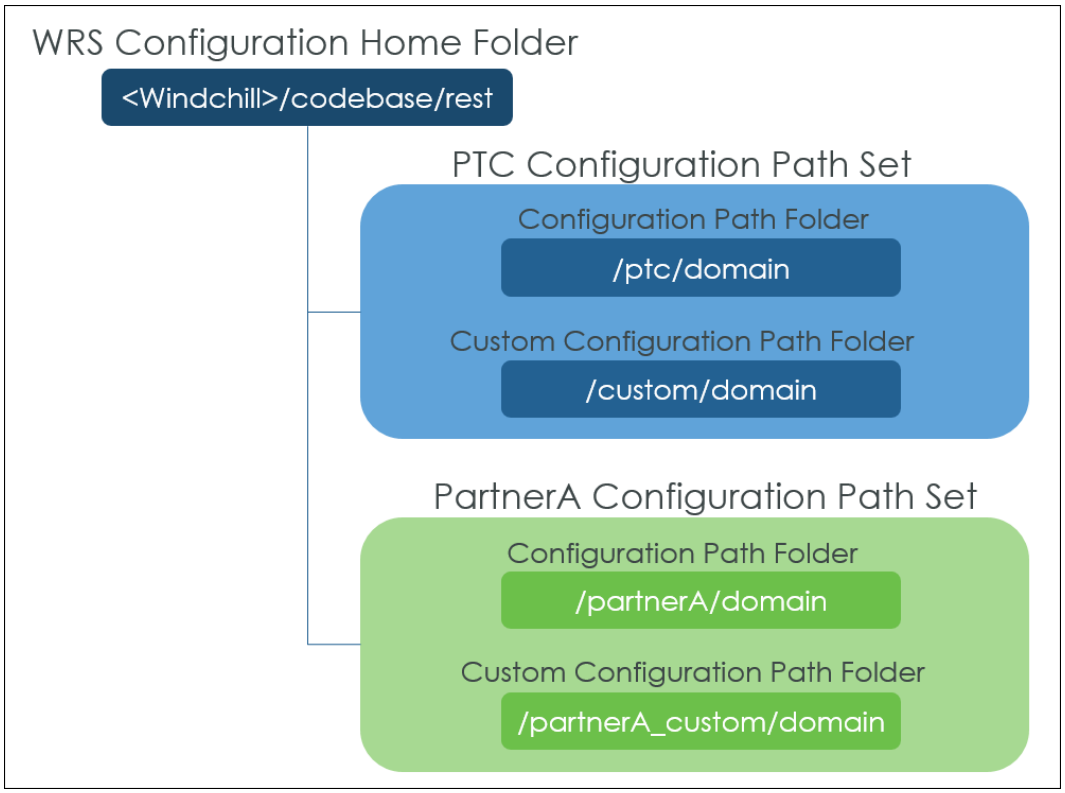
A configuration path set is specific to a domain provider who provides domain configurations for Windchill. PTC is the primary domain provider. However, PTC partners and PTC Customer Success organization, can act as domain providers, if they create and publish domain configurations independent of the release of Windchill REST Services.

The configuration path set for a provider has two types of folders:

- Configuration path folder
- Custom configuration path folder

The configuration path folder for a given provider contains configurations files provided by the domain provider. The custom configuration path folder contains configuration files created by customers which extends the entity definitions available in the configuration path folder. The configuration files available in the

configuration path folder of a domain provider must not be modified. You can create new configuration files under the custom configuration path folder of the domain provider.



Windchill REST Services reads all the configuration path sets in the configuration home to provide a unified view of the domains available in a Windchill installation. For each configuration path set, Windchill REST Services consolidates the configuration files from both configuration path and custom configuration path folders to create a unified list of domains and their EDM available for a provider.

For domains provided by PTC, the configuration path and the custom configuration path folders are:

- PTC Configuration Path—`<Windchill>/codebase/rest/ptc/domain/`, where `<Windchill>` is the Windchill installation directory.
  - This path is reserved for domain configurations that are provided by PTC. The domains and entities are installed at this path.
  - Do not modify the files at this path.

    - You must not create any new configuration files at this location as future updates from PTC will delete and recreate files in this path.
- PTC Custom Configuration Path—`<Windchill>/codebase/rest/custom/domain/`, where `<Windchill>` is the Windchill installation directory.
    - This path is provided for custom configuration files.
    - By creating custom configuration files at this location, customizers can extend PTC-provided domains, or create new custom domains.

> **Note**
>
> The configuration details and examples in this guide have been explained with PTC as the domain provider. The details explained for PTC domain provider can be applied to domain configurations created by PTC partners and customers in their own configuration path sets.

Configuring a Domain

To configure a domain create the following folder structure along with the required JSON files at the custom configuration path:

`<Windchill>/codebase/rest/custom/domain`

- `<Domain Folder>`
    - `<Version Folder>`
        - `complexType`
        - `entity`
        - `import.json`
        - `import.js`
- `<Domain JSON File>`

While configuring a domain, create the `<Domain Folder>`. The name of folder is the name of the domain identifier. The domain identifier of the domain is the name specified by customizers in the `id` property in the `<Domain JSON File>`.

The `<Version Folder>` is the name for the version of the domain API. There can be multiple version folders under the domain folders, each representing the domain configuration for that version. These folders are named `v1`, `v2`, `v3` and so on. The `<Version Folder>` contains subfolders that contain the configuration files for entities, complex types, and configuration for domain imports.

The folder `complexType` contains configuration files for OData complex types. If your domain contains complex types, this folder will contain a `.json` file for each complex type defined in the domain. In OData, complex types are structures of primitive types, and are used to combine related properties. For example, the PTC domain defines a complex type called `EnumType` that combines two string properties, `Value` and `Display`. The `EnumType` complex type is used to represent a Windchill enumeration type. `Value` represents the property value persisted in the database. `Display` represents the localized property value used for display.

The `entity` folder contains two configuration files, a `.json` and a `.js` file, for each entity in the domain. The `.json` file specifies the properties of the entity being configured and the `.js` file contains its hook implementations.

The `import.json` file specifies the other domains that are imported into the domain being configured. Importing other domains in a domain is an OData capability that allows EDM of the imported domains to be used in the referencing domain. For example, PTC Common domain, which is a domain provided by PTC in Windchill REST Services, is imported by all other domains. It contains common constructs, such as, common complex types `EnumType`.

The `import.js` file contains implementations for unbound functions and actions. This file is needed only if unbound actions and functions are defined in the domain.

The `<Domain JSON File>` is a JSON file with the same name as the domain identifier of the domain, and has .json specified as extension in its file name. For example, for the Product Management domain, this file is called `ProdMgmt.json`. The `<Domain JSON File>` contains the metadata configuration for the domain. This file contains properties such as, domain name, domain identifier, and so on.

Domain JSON File

The `<Domain JSON File>` is a JSON file with the same name as the domain identifier, and has `.json` specified as extension in its file name. For example, for domain identifier ProdMgmt, the `<Domain JSON File>` file name is ProdMgmt.json. The file contains configuration metadata for the domain. The configuration metadata is specified in a JSON object with the following properties:

- *name*—Name of the domain. For example, Product Management.
- *id*—An unique identifier of the domain in camel case. For example, ProdMgmt for Product Management domain.
- *description*—Description of the domain.
- *namespace*—An OData identifier that appears in the domain EDM as a namespace qualifier for a domain. For example, PTC.ProdMgmt.

- *containerName*—An OData identifier that appears in the domain EDM as a container for the entity sets of the domain.
- *defaultVersion*—Default version of the domain API that is returned to the clients if they do not request a specific version of the domain API. The values for this property are specified as 1, 2, 3 and so on in the JSON file. The framework interprets the value of 1 as `v1`. It searches for the `<Domain Folder>/v1` folder for entity configurations that must be used for processing requests. Similarly, a value of 2 is interpreted as `v2`, and so on.

For example, the `ProdMgmt.json` file from the Product Management domain is as shown below:

```
{
  "name":"Product Management Domain",
  "id":"ProdMgmt",
  "description":"PTC Product Management Domain",
  "namespace":"PTC.ProdMgmt",
  "containerName":"Windchill",
  "defaultVersion":"1"
}
```

Importing JSON File

The `import.json` file is used to specify the domains that are imported by the domain being configured. The imported domains are specified in the `imports` property, which is a collection of JSON objects. Each object is of the form:

```
{name="<domain name>", version="<domain version>"}
```

where,

- `<domain name>` is the name of the domain being imported
- `<domain version>` is the version of the domain being imported

An example of `import.json` for the Product Management domain. The file shows that the domains PTC, DataAdmin, DocMgmt and PrincipalMgmt with version 1 are being imported into the Product Management domain:

```
{
 "imports":[
   {name="PTC", version="1"},
   {name="DataAdmin", version="1"},
   {name="DocMgmt", version="1"},
   {name="PrincipalMgmt", version="1"}
  ],
  "functions":[],
  "actions":[]
}
```

Versioning of the Domain API

Windchill REST Services supports versioning of the APIs provided by a domain. The domain configurations are defined in version specific folders, such as, `v1`, `v2`, `v3`, and so on.

Clients can request a specific version of a domain resource in the URL. For example, the URL to request version 1 of the Product Management domain is:

```
https://windchill.ptc.com/Windchill/servlet/odata/v1/ProdMgmt/
```

Alternately, the version can be specified in the `Accept` header of the HTTP request. For example, to request version 1 of the Product Management domain use the URL:

```
https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/
```

Specify the version in the `Accept` header as:

```
application/vnd.ptc.api+json;version=3
```

You must specify the version only if clients need a specific version for backward compatibility. If not, it is recommended that version must not be specified in the URL or `Accept` header. The server must send the default version of the API.

Configuring Unbound Functions

The framework supports configuring unbound functions in a domain. In OData protocol, an unbound function is considered to be an operation that does not change the state of a service. The framework treats an unbound function as a read-only operation available in a domain. After the unbound function is configured, it is invoked by a GET request to the URL:

```
<Domain Root>/<Unbound Function Name>(<param1>=<value1>, <param2>=<value2>)
```

To configure an unbound function, perform the following steps:

1. Specify the properties of the function: name, input parameters, and return type.
2. Define the implementation logic for the function.

The properties of an unbound function are specified in the `import.json` file of the domain. In the file, under `functions`, specify the following properties:

- *name*—Name of the unbound function. The function is invoked from the URL with this name.
- *importName*—Name of the import operation.
- *description*—Description of the function.
- *includeInServiceDocument*—This is applicable for unbound functions. Defines if the function can be requested as a service in the container. The default value is set to false.

- *parameters*—A collection of parameters to be passed to the function. You can specify multiple parameters separated by commas. Specify the following parameters for a function:
  - *name*—Name of the parameter.
  - *type*—Type of the parameter. The parameter can be a primitive or an entity type. If the parameter is an entity type, the value is specified in the URL to the entity.
  - *isNullable*—Specifies if a property can be set as null. The default value is set to false.
  - *isCollection*—Specifies if the property represents a collection. The default value is set to false.
- *returnType*—Information about what the function returns. Specify the following properties for the return types:
  - *type*—Type of object that is returned. The return type can be a primitive or an entity type.
  - *isNullable*—Specifies if a property can be set as null. The default value is set to false.
  - *isCollection*—Specifies if the property represents a collection of objects or entities. The default value is set to false.

For example, consider a configuration file `import.json` for an unbound function `GetEndItems`. The function returns a collection of Part entities, and takes no input parameters:

```
{
  "imports:[
     {name="PTC", version="1"},
     {name="DataAdmin", version="1"}
  ],
  "functions":[{
     "name": "GetEndItems",
     "description": "Gets a list of end items parts",
     "parameters": [],
     "returnType": {
        "type": "Part",
        "isCollection": true
     }
  }]
"actions":[]
}
```

An unbound function is implemented in the `import.js` file. The function is implemented as below:

- The function name starts with `function_`, followed by the name of the function that is defined in the `import.json` file.
- The function takes two input parameters:
  - *data*—The parameter is of type `FunctionProcessorData`, which is a data structure, that contains information about the processing logic of the framework. This information can be used while implementing the function.
  - *params*—The parameter is of type `Map<String, Parameter>` and it contains the hashmap of input parameter names and values passed by the client to the function. The name of the input parameter is the key in the map and the parameter value passed by the client is the value for the key.

Configuring Unbound Actions

The framework supports configuring unbound actions in a domain similar to unbound functions. An unbound action is considered as an operation that can change the state of a service. Due to this, the framework supports calling an action with a POST request. After the unbound action is configured, it is invoked by a POST request to the URL `<Domain Root>/<Domain Namespace>.<Unbound Action Name>`. The body of the POST request contains the parameters that will be passed to the action.

An unbound action is configured in the `import.json` file in the same way as an unbound function except the following differences:

- An unbound action is specified in the `actions` collection property in the `import.json` file.
- While specifying the action in the `import.json` file, the property `includeInServiceDocument` is not applicable to actions. This is because actions cannot be included in the service document available at the domain root.
- The action names defined in the `import.json` file start with `action_`.

Configuring Entities in a Domain

Entities available in domains are configured by creating the following two files in the `entity` folder:

- `<Entity JSON>`
- `<Entity JS>`

The name of the `<Entity JSON>` file is the plural of the entity name, and has `.json` specified as extension in its file name. For example, `Parts.json` contains the configuration of the entity `Part`. Similarly, `<Entity JS>` file is the plural of the entity name, and has `.js` specified as extension in its file name.

The `.json` file specifies the structural properties, navigation properties, inheritance of Windchill functionality, bound functions, and bound actions of an entity. The `.js` file contains JavaScript implementation of the bound actions and functions of an entity, and also contains implementation of hooks provided by customizers to override or enhance framework processing logic for the entity.

Basic Information for Configuring Entities

To configure an entity, the framework requires information on the following entity properties:

- *name*—Name of the entity. Entity name is always defined in camel case. For example, `WindchillPart`.
- *collectionName*—Name of the entity collection. For example, Parts.
- *type*—Type of entity. Set the value as `wcType` for entities that are backed by Windchill types. For other entities specify the value as `basic`.
- *wcType*—This property must be set if *type* property is specified as `wcType`. The entity type is backed by Windchill types. For example, `wt.part.WTPart`.

> **Note**
>
> If you configure `wctype as [domain].<Windchill_type>` in the `<Entity JSON>` file, then `[domain]` is read as the internet domain, which is set for the exchange container. For example, if the internet domain for the exchange container is set to `com.ptc` and `wctype` for `DynamicDocument` is configured as:
>
> `"wcType": "[domain].DynamicDocument"`
>
> The `wctype` is read by clients as `com.ptc.DynamicDocument`.

- *description*—Description of the entity type.
- *operations*—List of CRUD operations that are permitted on entities. For `wcType` entities, the READ operation is permitted by default. The other operations that are permitted must be specified explicitly.

For basic entities, the operations that are permitted must be explicitly specified in the configuration file.

- *hasCommonProperties*—Indicates if an entity contains Windchill attributes that are common for objects. For example, attributes such as, `Name`, `Number`, and so on. When the property is set to true for an entity, the action `UpdateableViaAction` is available in the domain. This action is used to edit the values of common attributes.

> **Note**
>
> If you specify *hasCommonProperties* as `true`, it is mandatory to specify at least one attribute as *common*. See the section Configuring Structural Properties on page 42, for more details.

Configuring Structural Properties

Structural properties in OData are the attributes that define a business object or entity. Windchill REST Services reads the properties of attributes, which is a JSON array, from the `<Entity JSON>` file. Each entry in the attribute defines one structural property for the entity being configured. The structural property comprises of the following parameters:

- *name*—Name of the structural property. Property name is always defined in camel case.
- *internalName*—Internal name of the Windchill property that corresponds to the structural property being configured.
- *type*—Framework data type for the structural property being configured. Framework data type is same as the OData primitive type without the Edm prefix. For example, the framework data type corresponding to OData `Edm.Double` is `Double`.
- *required*—Specifies if the attribute is required. Set the parameter value as true or false.
- *readOnly*—Specifies if the attribute is read only. Set the parameter value as true or false.
- *driver*—Specifies if the attribute is a driver attribute. Set the parameter value as true or false.
- *nonFilterable*—Specifies if the attribute can be queried using a `$filter` expression. Set the parameter value as true or false.

  When you set the parameter value as true, the attribute does not support `$filter` expressions. The attribute is added to the `FilterRestrictions` annotation in the entity set of the entity. The

`FilterRestrictions` annotation is defined in the Capabilities Vocabulary of OData, which specifies properties and attributes that do support `$filter` expressions.

If such an attribute is used in a `$filter` expression, the server returns an error message.

- *isCollection*—Specifies if the property supports multivalued attributes. Set the parameter value as true or false.

  When you set the parameter as true, the property supports multivalued attributes, such as, complex types.
- *common*—Specifies if the parameter is a Windchill attribute that is common for objects and can be edited. Set the parameter value as `true` to make the attribute editable.

Configuring Navigation Properties

Navigation properties in OData are the reference attributes of an entity that points to another entity. By default, navigation properties are not available on an entity representation when it is accessed by an OData client. The OData client must expand these properties explicitly if they want the associated entities to be available when the entity is being accessed. Windchill REST Services reads the navigation property, which is a JSON array, from the `<Entity JSON>` file. Each entry in the `navigations` section defines one navigation property for the entity being configured. The navigation property consists of the following parameters:

- *name*—Name of the navigation property. Property name is always defined in camel case.
- *target*—OData entity set which is the target of this navigation. The entity being configured is the source entity. The entity set to which the navigation is being configured is the target entity.
- *type*—OData type of the target entity set.
- *isCollection*—Boolean which checks if the navigation to the target entity results in an entity set.
- *containsTarget*—Boolean which checks if the navigation property is a containment navigation property. A containment navigation property in OData enables the read URL of a navigation property to be implicitly treated as an entity set.
- *traversal*—A traversal string for navigation between two entities. You must create alias attribute mapping. You can navigate from one Windchill object type to another using the alias attribute mapping. Search for alias attribute mapping in the Windchill help center.

  Use this parameter to apply a filter on a navigation property. This parameter is mandatory if you want to filter entity sets.

For example, consider the `Parts.json` file, where *traversal* is defined for the `Uses` navigation property. In this case, you can apply the filter on the `Uses` navigation property to filter the `Parts` entity set.

```
{
   "name": "Uses",
   "target": "PartUses",
   "type": "PartUse",
   "isCollection": true,
   "containsTarget": true,
"traversal":"usedBy@wt.part.WTPartUsageLink"
 }
```

See the section Support for $filter on Navigation Properties on page 21 for examples on how to use `$filter` on navigation properties.

Configuring Bound Functions

The framework supports configuring functions that are bound to entities. A function bound to an entity is invoked on an instance of the entity. In OData protocol, an unbound function is considered to be an operation that does not change the state of an entity instance on which it is invoked. After the bound function is configured, it is invoked by a GET request to the URL:

```
<Domain Root>/<Entity Set>(<key>)/<Bound Function Name>(<param1>=<value1>,
<param2>=<value2>)
```

A bound function is configured in the same way as an unbound function except the following differences:

- A bound function is specified in the `functions` collection property in the `<Entity JSON>` file.
- While specifying the function in the `<Entity JSON>` file, the property `includeInServiceDocument` is not applicable to bound functions. This is because bound functions cannot be included in the service document available at the domain root.
- The first parameter in the function specification is called the binding parameter. The parameter must be of the same type as the entity that is bound to the function.
- The function names defined in the `import.js` file start with `function_`.

Configuring Bound Actions

The framework supports configuring actions that are bound to entities. An action bound to an entity is invoked on an instance of the entity. OData protocol considers a bound action to be an operation that changes the state of an entity instance on which it is invoked. After the bound action is configured, it is invoked by a POST request to the URL:

```
<Domain Root>/<Entity Set>(<key>)/<Domain Namespace>.<Bound Action Name>
```

The body of the POST request contains the parameters that must be passed to the action.

A bound action is configured in the same way as an unbound action except the following differences:

- A bound action is specified in the `actions` collection property in the `<Entity JSON>` file.
- While specifying the action in the `<Entity JSON>` file, the property `includeInServiceDocument` is not applicable to bound actions. This is because bound actions cannot be included in the service document available at the domain root.
- The first parameter in the action specification is called the binding parameter. The parameter must be of the same type as the entity that is bound to the action.
- The action names defined in the `import.js` start with `action_`.

Inheriting Windchill Capabilities

Windchill provides a capability called Workable for its business objects. Windchill persistables which implement this capability have certain attributes such as `CheckoutState`. Further, persistables which implement Workable can be checked out and checked in.

The framework supports the Workable capability of Windchill and allows entities being configured to inherit this capability. An entity that inherits Workable automatically inherits the structural property `CheckoutState` without having to explicitly configure it in the `<Entity JSON>` file. Also, the entity inheriting Workable automatically gets bound actions such as, CheckIn, CheckOut, and UndoCheckOut without having to explicitly define them.

Workable is one of the capabilities supported by the framework. The complete list of Windchill capabilities supported by the framework is shown in the following table:

Windchill Capability	Inheriting Entity Behavior
`versioned`	Entity properties are automatically available in the EDM: • VersionID—Version identifier of the entity • Revision—Revision of the entity • Version—Version of the entity • Latest—Checks if the entity is the latest version Entity navigation properties are automatically available in the EDM: • Versions—Collection of all entity versions • Revisions—Collection of latest iteration of each entity revision Bound actions are automatically available in the EDM: • Revise—Revises the entity when called
`contextManaged`	Entity navigation properties are automatically available in the EDM: • Context—Supports navigation to a Container
`lifecycleManaged`	Entity properties are automatically available in the EDM: • LifeCycleTemplateName • State
`viewManageable`	Entity properties are automatically available in the EDM: • View
`workable`	Bound actions are automatically available in the EDM: • CheckOut—Checks out the entity • CheckIn—Checks in the entity • UndoCheckOut—Undo an entity checkout • IsCheckoutAllowed—Checks if checkout is allowed on an entity
`representable`	Entity navigation properties are automatically available in the EDM:

Windchill Capability	Inheriting Entity Behavior
	• Representations—Supports navigation to a viewable representation from the entity
`organizationOwned`	Entity navigation properties are automatically available in the EDM: • Organization—Supports navigation to the organization principal of the entity Entities which are created within an organization context are automatically owned by the organization participant associated with the organization context. From Windchill REST Services 1.2 and later, if the preference **Expose Organization** in the **Preference Management** utility is set to `Yes` in Windchill, you can specify a different organization when you create parts or documents. Organization is access-controlled. A minimum access level of read is required to specify a different organization. In the request body of the URL, specify the OID of the organization or CAGE code in the `Organization@odata.bind` annotation. If the preference **Expose Organization** is set to `No`, you cannot specify a different organization. If you try to specify a different organization, an error message appears.
`foldered`	Entity properties are automatically available in the EDM: • FolderName—Folder where the entity is located • CabinetName—The cabinet where the folder lives • FolderLocation—The path to the folder

Windchill Capability	Inheriting Entity Behavior
	Entity navigation properties are automatically available in the EDM: • Folder—Supports navigation to the folder of the entity
`subtypeable`	Entity subtypes are automatically available in the EDM of the domain. If you add new subtypes in Windchill, the subtypes are available in the entity. The attributes available in the subtype are also available as structural properties of the entity in the EDM of the domain. **Note** The subtypes that are excluded in the `<Entity JSON>` file under `wcExcludedTypes`, are not available as subtypes. If you define a subtype in the `<Entity JSON>` file and exclude it in `wcExcludedTypes`, the subtype is still available in the EDM. See Excluding Subtypes of Enabled Windchill Types on page 53Excluding Subtypes of Enabled Windchill Types on page 53 for more information. When the subtypes are available in Windchill REST Services, they follow a standard naming convention. See Naming Convention for Subtypes on page 52, for more information. In the following domains provided by PTC, `subtypeable` is available for the following entity types: • PTC Document Management domain—`Document` entity. • PTC Product Management domain—`Part` entity.

Windchill Capability	Inheriting Entity Behavior
	• PTC Change Management domain—`ProblemReport`, `ChangeNotice`, `ChangeRequest`, and `Variance` entities. • PTC CAD Document Management domain—`CADDocument` entity. • PTC Manufacturing Process Management domain—Entities, such as, `Operation`, `OperationUsageLink`, and so on. See the section subtypeable and softattributable Attributes on page 122, for more information. • PTC Regulatory Master domain—`RegulatorySubmission` entity.
`softattributable`	Attributes of entity types and its subtypes are automatically available in the EDM of the domain. If you add new attributes in a type or subtype in Windchill, the attributes are available as structural properties of an entity in the EDM of the domain. In the following domains provided by PTC, `softattributable` is available for the following entity types: • PTC Document Management domain—`Document` entity. • PTC Product Management domain—`Part` entity. • PTC Change Management domain—`ProblemReport`, `ChangeNotice`, `ChangeRequest`, and `Variance` entities. • PTC CAD Document Management domain—`CADDocument`, `CADDocumentUse` and `CADDocumentReference` entities.

Windchill Capability	Inheriting Entity Behavior
	• PTC Manufacturing Process Management domain—Entities, such as, `Operation`, `OperationUsageLink`, and so on. See the section subtypeable and softattributable Attributes on page 122, for more information. • PTC Regulatory Master domain—`RegulatorySubmission` entity.
`classifiable`	Entities that inherit `classifiable`, automatically get the property of type `ClassificationInfo`. This property is used to classify the entity.

Windchill Capability	Inheriting Entity Behavior
`securityLabeled`	Security labels associated with the Windchill objects are automatically available in the EDM of the entity. All the security labels are added as properties of the entity. These properties are annotated with `PTC.SecurityLabel` and `PTC.ReadOnly`. See the section PTC Annotations on page 27, for more information on annotations. Windchill 11.2.0.0 and later support multiple values for a security label. The `securityLabeled` capability also supports multiple values for security labels. If a security label has multiple values, the response returns a comma-separated string of values. In the following domains provided by PTC, `securityLabeled` is available for the following entity types: • PTC Document Management domain—`Document` entity. • PTC Product Management domain—`Part` entity. • PTC Change Management domain—`ProblemReport`, `ChangeNotice`, `ChangeRequest`, and `Variance` entities.

For an entity to inherit one of these capabilities, edit the `<Entity JSON>` file and add the following entry in the collection for `inherits` property:

```
{“name”: “<Windchill Capability>”}
```

The following example shows how an entity inherits `versioned` and `workable` capabilities:

```
{
…
“inherits”: [
    {
        “name”: “versioned”
     }, {
        “name”: “workable”
```

```
      }
]
…
}
```

Naming Convention for Subtypes

The `subtypeable` capability enables subtypes, which are available in Windchill, in the some domains in Windchill REST Services.

When the subtype is enabled in Windchill REST Services, the following naming convention is used:

- The subtype created in Windchill appears in Windchill REST Services with its internal name.
- The domain name which is added as prefix to the internal name is removed.
- The first character of the internal name is capitalized.

  For example, consider a parent entity `Part` in Windchill. Under `Part`, add a new subtype with the display name `demo`. The domain name of the subtype is `com.ptc`. The internal name of the subtype in Windchill is `<DomainName.SubtypeName>`. In this example, the internal name of the subtype is `com.ptc.demo`. When the subtype is available in Windchill REST Services, its name is `Demo`.
- In Windchill REST Services, if subtypes have duplicate internal names, then an error message is returned for the duplicate name. You must change the internal name of the subtype in Windchill to resolve the duplicate name issue in Windchill REST Services.

  For example, consider a parent entity `Part` in Windchill. Under `Part`, two subtypes are available with the display names `demo-assembly` and `demo_assembly`. In Windchill REST Services, you cannot use characters, such as, a hyphen (-) and an underscore (_) in internal names. In this case, the name of both the subtypes in Windchill REST Services is `Demoassembly`. In Windchill REST Services, the first subtype is added with internal name `Demoassembly`. For the second subtype, the duplicate name error is returned.
- If you define a subtype with an internal name in the `<Entity JSON>` file, and create a subtype in Windchill with the same internal name, then Windchill REST Services gives precedence to the subtype created in `<Entity JSON>` file.

Excluding Subtypes of Enabled Windchill Types

When an entity is configured with a Windchill type, any subtypes of the Windchill type are also included in the output of the entity queries. The framework enables you to exclude Windchill subtypes from being mapped to entities. To do this, add the following entry in the <Entity JSON> file:

```
"wcExcludedTypes":[“<Windchill Subtype 1>”, “<Windchill Subtype 2>”, …]
```

For example, consider a WTDocument with subtypes Agenda and Plan. If you want to exclude these subtypes when the Document entity is created, add the following entry in the JSON file:

```
“wcExcludedTypes”: [“org.rnd.Agenda”, “org.rnd.Plan”]
```

Note

If you define a subtype in the <Entity JSON> file and exclude it in wcExcludedTypes, the subtype is still available in the EDM.

Disabling Entity Set for an Entity in the Service Document

When an entity is configured, by default the entity set of that entity is available in the service document of the domain. Customizers can choose to remove the entity set from the service document. To do this, set the property includeInServiceDocument to false in the <Entity JSON> file. When you remove an entity set from the service document, the entity set is hidden from the clients that use the service document.

Processing HTTP Requests for OData URLs

The framework provides the default processing for HTTP requests made by clients to OData URLs.

OData URLs for EDM such as, https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/$metadata, are processed by Domain Provider, Entity Provider, and Entity Delegate classes. The Domain Provider and Entity Provider classes read and process the configuration files. The Entity Delegate classes create the metadata response for entities in the domain.

OData URLs for entities and entity sets are processed by entity processor classes. The framework provides two types of processor classes, BasicEntityProcessor and PersistableEntityProcessor.

The `BasicEntityProcessor` class is used to process requests for entities and entity sets that are not mapped to Windchill persistables.

The `PersistableEntityProcessor` class is used to process requests made to entities and entity sets that are mapped to Windchill persistables.

A GET request to the URL for an entity set is processed by the `PersistableEntityProcessor` class. For example, consider a GET request to the URL `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts`, which is processed by the `PersistableEntityProcessor`. While processing a GET request, the default processing logic of the framework reads the persistable objects of the mapped type, in this case `WTParts` from Windchill. Each object is converted into relevant OData entity. The framework then returns the entity set in the format requested by the client.

A GET request to the URL for an entity is also processed by the `PersistableEntityProcessor` class. For example, consider a GET request to the URL `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:87676')`, which is processed by the `PersistableEntityProcessor`. In this case, the default processing in the framework reads the specific persistable object identified by the object reference in the URL, converts it to an entity, and then returns the entity representation in the requested format.

While processing a POST request on an entity set URL, the framework takes the entity representation provided in the POST body in the specified format, converts it into a Windchill persistable, and saves it to Windchill.

PATCH request to an entity URL reads the persistable, changes it, and then sends the updated representation in the response based on the requested format. DELETE request works similar to the PATCH request.

The framework enables customizers to override or enhance the default processing. When processing entity requests the framework searches for JavaScript implementation of hooks in the `<Entity JS>` file of the entity being processed. If hooks are found, then the framework executes the code they contain. Depending on the value returned by the hooks, the framework either continues its default processing or abandons it.

The hooks available in the framework are explained below:

`Object readEntityData(EntityProcessorData processorData)`

The framework calls this hook when the client sends a GET request to an entity instance. The hook is used to read the backing object from the store. The backing object is converted to an entity and returned to the client.

For basic REST entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, the hook is optional. The framework layer of Windchill REST Services provides a default implementation that uses Windchill APIs to read the persistable object from Windchill. If the hook is not implemented, then framework layer uses the default implementation to read the pesistable object. When the hook is implemented, it returns the backing persistable object for the entity to the framework. If the hook sets the `continueProcessing` flag to `true` in `processorData`, the object returned by the hook to the framework is ignored. In this case, the framework continues reading the backing object using the default implementation.

`Collection<Object> readEntitySetData(EntityProcessorData processorData)`

The framework calls this hook when the client sends a GET request to an entity set. The hook is used to read the collection of backing objects that belong to an entity set. The collection of backing objects is converted to an entity set and returned to the client.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, it is recommended not to override this hook as the framework provides a default implementation, which is optimized for features and performance. In its default implementation, the framework uses Windchill search APIs to search for objects in the entity set while honouring the `$filter`, `$top`, `$skip` parameters and server generated paging.

If you override the hook, you must provide performant implementation that honours the OData query parameters `$filter`, `$top`, `$skip` and server generated paging. When the hook is implemented it returns a collection of backing persistable objects. If no objects are found, the hook returns `NULL`.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the object collection returned by the hook is ignored. In this case, the framework uses the default implementation to read backing persistables, which belong to the entity set. It converts the persistable collection to an entity set, and returns the entity set to the client.

`Map<Object, Collection> getRelatedEntityCollection(NavigationProcessorData processorData)`

The framework calls this hook while navigating from source to target entities.

For both the basic REST and Windchill entities, this hook must be implemented for the navigation properties, which are explicitly specified in `navigations` section of the JSON configuration file of an entity. You must use Windchill service APIs to perform the navigation between persistables. It is recommended not to use the lower level persistence APIs. When you use the service APIs, it ensures that the business rules implemented in Windchill are honoured.

When the hook is implemented, it returns a map. The keys of the map are the source objects. The value of each key is a collection of target backing objects, which are persistable objects for Windchill that are obtained by navigating from the source.

`Entity createEntityData(Entity entityToCreate, EntityProcessorData processorData)`

This hook contains the complete logic for storing a persistable object in the data store. The storage begins with the entity that is sent to Windchill REST Services in the body of a POST request.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, the hook must not be implemented. Windchil REST Services provides a default implementation of the hook which:

1. Converts the entity that is passed in the body of the POST request to a persistable object
2. Stores the persistable object in Windchill
3. Retrieves the object from Windchill
4. Converts the object to an entity
5. Returns the entity representation to the caller

During the create process, the default implementation provides more granular hooks such as, `operationPreProcess`, `storeNewObject`, and `operationPostProcess`. Customizers should implement these granular hooks instead of the `createEntityData` hook, which represents the overall create process.

If the hook is implemented, it returns the entity that was created back to the framework. If the hook sets the `continueProcessing` flag to `true`, the entity returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Entity updateEntityData(Entity entityToUpdate, EntityProcessorData processorData)`

This hook contains the complete logic for updating a persistable object in the data store. The update begins with the entity that is sent to Windchill REST Services in the body of a PATCH request.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, the hook must not be implemented. Windchil REST Services provides a default implementation of the hook which:

1. Reads the backing persistable object for the entity being updated
2. Changes those properties that are specified in the PATCH request
3. Stores the persistable object in Windchill
4. Retrieves the newly saved object from Windchill
5. Converts the object to an entity
6. Returns the entity representation to the caller

During the update process, the default implementation provides more granular hooks such as, `operationPreProcess`, `saveObject`, and `operationPostProcess`. Customizers should implement these granular hooks instead of the `updateEntityData` hook, which represents the overall update process.

If the hook is implemented, it returns the entity that was updated back to the framework. If the hook sets the `continueProcessing` flag to `true`, the entity returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`void deleteEntityData(Entity entityToDelete, EntityProcessorData processorData)`

This hook contains the complete logic for deleting an entity. The logic includes finding and deleting the corresponding backing object in the data store for the entity being deleted.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, the hook must not be implemented. Windchil REST Services provides a default implementation of the hook which calls other more granular hooks, such as, `operationPreProcess`, `deleteObject`, `operationPostProcess`, and so on, when an entity is deleted. Customizers should implement these granular hooks instead of the `deleteEntityData` hook.

When the hook is implemented, it does not return anything back to the framework.

If the hook sets the `continueProcessing` flag to `true`, Windchill REST Services continues the default processing as if the hook was never implemented.

`Map<Entity, Object> toObjects(Collection<Collection>, entitiesToConvert, EntityProcessorData processorData)`

The framework calls this hook when it is converting entities to backing objects. For example, when clients create, update, delete, navigate, or expand entities.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, the framework provides a default implementation. The implementation uses the information from the request payload to construct the objects in memory. When the hook is implemented, it returns a map. The keys of the map are entities, and the value for each key is the backing object that corresponds to the key entity. If the hook sets the `continueProcessing` flag to `true` in `processorData`, the map returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Map<Object, Entity> toEntities(Collection<Object>objectsToConvert, EntityProcessorData processorData)`

The framework calls this hook when it is converting backing objects to entities. For example, during the read, create, and update requests, navigations or expansions.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, the framework provides a default implementation. The implementation uses the information in the entity configuration to construct the entities from the objects.

When the hook is implemented, it returns a map. The keys of the map are objects, and the value for each key is the entity that corresponds to the key object.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the map returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Boolean isValidEntityKey(String key, EntityProcessorData processorData)`

The framework calls this hook to validate the key (generally the property ID) of the entity. It is called during read, update or delete operations of an entity.

For basic entities, the implementation of the hook is optional. If the hook is not implemented, any key is considered valid.

For Windchill entities, there is default implementation provided in case the hook is not implemented.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the value returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Boolean isValidNavigation(String name, Object sourceObj, String id, EntityProcessorData processorData)`

The framework calls this hook when navigating from source to target entities. The implementation of this hook should check if the navigation from a source object to a target is valid. The implementation returns:

- `true` if the navigation is valid
- `false` if the navigation is invalid
- `null` if the navigation is not defined in the JSON file

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, this hook must be implemented for any navigation property specified in the entity JSON file.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the value returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Collection<AttributeData> processAdditionalAttributes(Entity entity, Object entityObject, EntityProcessorData processorData, PersistableEntityProcessor)`

This hook is called by the framework when it has built a persistable and has not committed it to the database. The hook allows customizers to add and change attribute values on the persistable before committing it.

`Object operationPreProcess(Object object, Entity entity, EntityProcessorData processorData, PersistableEntityProcessor)`

The framework calls this hook during the create or update operation of an entity. The hook is called when the entity is converted to an object just before the create or update operation is performed on the object. The hook contains preprocessing logic for the object before the actual operation takes place. If the hook implementation throws the exception `OdataApplicationException`, the operation is aborted, and an error is returned to the caller.

The implementation of the hook is optional.

When the hook is implemented, it returns the object that must be used to execute the operation. If the hook returns a null value, the operation is aborted, and an error is returned to the caller.

`void operationPostProcess(Object object, Entity entity, EntityProcessorData processorData, PersistableEntityProcessor)`

The framework calls this hook during the create or update operation of an entity, after the operation is complete. The hook contains postprocessing logic. This logic is called in the same transaction as the operation itself. If the hook implementation throws the exception `OdataApplicationException`, the transaction is rolled back, the operation is aborted, and an error is returned to the caller.

The implementation of the hook is optional.

`Object storeNewObject(Object, Entity, EntityProcessorData)`

This hook is called during the create operation of a new entity. It is called to store the new object to the data store after the entity has been converted to a backing object.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, if this hook is not implemented, the default implementation is used. The default implementation uses persistence manager to store the object to the data store. If you want use Windchill APIs to store the object to the data store, the hook must be implemented to override the default processing logic.

`void saveObject(Object entityObject, Entity entity, EntityProcessorData processorData)`

This hook is called during the update operation of an entity. It is called to save the object to the data store after the entity has been converted to a backing object.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, if this hook is not implemented, the default implementation is used. The default implementation uses persistence manager to save the object to the data store. If you want use Windchill APIs to store the object to the data store, the hook must be implemented to override the default processing logic.

If the hook implementation throws the exception `OdataApplicationException`, the operation is aborted, and an error is returned to the caller.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the value returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Collection<Object> storeNewObjects(Map<Entity, Collection> entitiesToObjects, EntityProcessorData processorData)`

This hook is called during the multiple create operation of new entities. It is called to store the new objects to the data store after the entities are converted to backing objects.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, if this hook is not implemented, the default implementation is used. The default implementation uses the persistence manager to store the objects to the data store. If you want to use other Windchill APIs to store the objects to the data store, the hook must be implemented to override the default processing logic.

If the hook implementation throws the `ODataApplicationException` exception, the operation stops, and an error message is returned to the caller.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the value returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Collection<Object> saveObjects(Map<Entity, Collection> entitiesToObjects, EntityProcessorData processorData)`

This hook is called during the multiple update operation of entities. It is called to save the objects to the data store after the entities are converted to backing objects.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, if this hook is not implemented, the default implementation is used. The default implementation uses persistence manager to save the objects to the data store. If you want to use other Windchill APIs to store the objects to the data store, the hook must be implemented to override the default processing logic.

If the hook implementation throws the `ODataApplicationException` exception, the operation stops, and an error message is returned to the caller.

If the hook sets the `continueProcessing` flag to `true` in `processorData`, the value returned by the hook is ignored. In this case, Windchill REST Services continues with the default processing as if the hook was never implemented.

`Map<Entity, Collection> multiOperationPreProcess(Map<Entity, Collection> entitiesToObjects, EntityProcessorData processorData, PersistableEntityProcessor entityProcessor)`

The framework calls this hook during the multiple create or update operation of entities. The hook is called when the entities are converted to objects just before the multiple create or update operation is performed on the objects. The hook contains preprocessing logic for the object before the actual operation takes place. If the hook implementation throws the `ODataApplicationException` exception, the operation is stopped, and an error message is returned to the caller.

The implementation of the hook is optional. When the hook is implemented, it returns the objects that must be used to execute the operation. If the hook returns a null value, the operation stops, and an error message is returned to the caller.

`void multiOperationPostProcess(List<Persistable> objectsToDelete, EntityCollection entities, EntityProcessorData processorData, PersistableEntityProcessor entityProcessor)`

The framework calls this hook during multiple delete operation of entities, after the operation is complete. The hook contains post processing logic. This logic is called in the same transaction as the operation itself. If the hook implementation throws an exception, the transaction is rolled back, the operation is aborted, and an error is returned to the caller.

The implementation of the hook is optional. When the hook is implemented, it does not return anything back to the framework.

`void deleteEntities(List<Persistable> objectsToDelete, OperationProcessorData processorData)`

This hook contains the complete logic to delete multiple entities. The logic includes finding and deleting the corresponding backing objects in the data store for the entities being deleted.

For basic entities, domain authors must provide implementation of this hook. The framework does not provide implementation of this hook in its default processing.

For Windchill entities, if this hook is not implemented, the default implementation is used. The default implementation uses the delete API of persistence manager to delete the objects from the data store. If you want to use other Windchill service APIs to delete multiple objects from the data store, the hook must be implemented to override the default processing logic. When the hook is implemented, it does not return anything back to the framework.

All the hooks accept either `EntityProcessorData` or `NavigationProcessorData` parameter objects as input.

- `NavigationProcessorData`—Contains information about source objects, target entity, entity set name, and so on which are required for navigating entity associations. See Summary Javadoc for NavigationProcessorData on page 244 for details.
- `EntityProcessorData`—Contains information which is required for processing a request to create, read, update, and delete entities. See Summary Javadoc for EntityProcessorData on page 247 for details.

Processing Batch Requests

Using batch requests, you can group multiple operations in a single HTTP request. Use the `$batch` attribute to request data and perform operations on the data.

For example, run the batch request as below:

```
https://windchill.ptc.com/Windchill/servlet/odata/<domain>/$batch
```

In a batch request, you can specify a series of individual batch requests or create change sets. Batch requests are represented as multipart MIME message. Specify the batch requests and change sets in relevant `Content-Type` header as distinct MIME parts. The requests are processed sequentially.

Individual batch requests support the following types of requests:

- Creating data
- Getting data
- Modifying data
- Invoking an action and function

If any of the individual batch requests from the series fail, the other batch requests are processed.

Change set is an atomic unit inside which you can define a set of requests. In a change set you define series of individual batch requests. However, if one or more individual batch requests from the series fail, the entire change set fails. In a change set, if the batch requests have modified any data before encountering a failed request, then all the data changes are rolled back. A change set has been implemented as a Windchill transaction.

Change set supports the following types of requests:

- Modifying data
- Invoking an action

Change sets do not support the GET operation.

After execution, batch requests return the appropriate HTTP response codes. The HTTP response body lists the response in the same order as the individual requests in the HTTP request body. However, the requests inside a change set may not be executed in the order specified in the change set.

From Windchill REST Services 1.3 onward, batch requests support references to entity and property values between requests and responses of different parts of a batch. A change set request can reference the property value of an entity from a change set of a previous batch request. To reference an entity property from the previous batch request, use the syntax `$<Content-ID_of_changeset>/<property_name>`. In the following example, `changeset2` references property `Name` from `changeset1`, which has `Content-ID` `1_1`.

```
POST /ProdMgmt/$batch
Content-Type: multipart/mixed;boundary=batch

--batch
Content-Type: multipart/mixed;boundary=changeset1

--changeset1
Content-Type: application/json
Content-ID: 1_1

POST ProdMgmt/Parts HTTP 1.1

{
  ... Part Entity Representation ...
}
--changeset1--

--batch
Content-Type: multipart/mixed;boundary=changeset2

--changeset2

Content-Type: application/json
Content-ID: 2_1

POST DocMgmt/Documents HTTP 1.1

{
  ...
  "Name": $1_1/Name,
  ...
}

--changeset2--
--batch--
```

A batch request can reference an entity from a change set of a previous batch request. To reference an entity from the previous batch request, use the syntax `$<Content-ID_of_changeset>`. In the following example, second `changeset` references entity from first `changeset`, which has `Content-ID 1_1`.

```
POST /ProdMgmt/$batch
Content-Type: multipart/mixed;boundary=batch

--batch
Content-Type: multipart/mixed;boundary=changeset

--changeset
Content-Type: application/json
Content-ID: 1_1

POST /ProdMgmt/Parts HTTP 1.1

{
  ... Part Entity Representation ...
}

--changeset
POST /ProdMgmt/Document HTTP 1.1

{
  ... Part Entity Representation ...
}

--changeset--
--batch

Content-Type: application/json
Content-ID: 2_1

POST $1_1/PTC.ProdMgmt.CheckOut HTTP 1.1

{
}

--batch--
```

Examples for Batch Requests

This section provides examples for batch requests

Batch Request for Creating Parts

The following is a batch request with change sets which creates parts.

```
POST /Windchill/servlet/odata/ProdMgmt/$batch HTTP/1.1
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

```
Content-Type: multipart/mixed;boundary=batch_request


--batch_request
Content-Type:multipart/mixed;boundary=changeset_abc

--changeset_abc
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-Id: 1

POST /Windchill/servlet/odata/ProdMgmt/Parts HTTP/1.1
Content-Type: application/json

{
"Name":"BatchPart1",
"Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:76625')"
}

--changeset_abc
Content-Type: application/http
Content-Transfer-Encoding:binary
Content-Id: 2

POST /Windchill/servlet/odata/ProdMgmt/Parts HTTP/1.1
Content-Type: application/json

{
"Name":"BatchPart2",
"Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:76625')"
}
--changeset_abc--

--batch_request--
```

The response to the batch request is as below:

```
--batch_3487ef0a-9598-41cb-a3cf-1d8616ed9e58
Content-Type: multipart/mixed; boundary=changeset_2f39c1fb-0e4b-4a60-9e2d-89cab451db7f

--changeset_2f39c1fb-0e4b-4a60-9e2d-89cab451db7f
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

HTTP/1.1 201 Created
OData-Version: 4.0
Content-Type: application/json;odata.metadata=minimal
Content-Length: 1149
```

{"@odata.context":"$metadata#Parts","ID":"OR:wt.part.WTPart:601168","Name":"BatchPart1",
"Number":"0000019103","EndItem":false,"TypeIcon":
{"Path":"https://windchill.ptc.com/Windchill/wtcore/images/part.gif","Tooltip":"Part"},
"Identity":"0000019103, BatchPart1, Demo Organization, A","GeneralStatus":null,
"ShareStatus":null,"ChangeStatus":null,"Supersedes":null,"AssemblyMode":
{"Value":"separable","Display":"Separable"},"DefaultUnit":{"Value":"ea","Display":"each"},
"DefaultTraceCode":{"Value":"0","Display":"Untraced"},
"Source":{"Value":"make","Display":"Make"},
"ConfigurableModule":{"Value":"standard","Display":"No"},"GatheringPart":false,
"PhantomManufacturingPart":false,"BOMType":null,"AlternateNumber":null,"View":"",
"CheckoutState":"Checked in","CheckOutStatus":"","Comments":null,
"State":{"Value":"INWORK","Display":"In Work"},"LifeCycleTemplateName":
"Basic","VersionID":"VR:wt.part.WTPart:601167","Revision":"A",
"Version":"A.0","Latest":true,"FolderName":null,"CabinetName":
"Default","FolderLocation":"/Default","OrganizationName":
"Demo Organization","CreatedOn":"2010-04-20T08:32:51Z",
"LastModified":"2010-04-20T08:32:51Z"}
--changeset_2f39c1fb-0e4b-4a60-9e2d-89cab451db7f
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

HTTP/1.1 201 Created
OData-Version: 4.0
Content-Type: application/json;odata.metadata=minimal
Content-Length: 1149

{"@odata.context":"$metadata#Parts","ID":"OR:wt.part.WTPart:601176",
"Name":"BatchPart2","Number":"0000019104","EndItem":false,"TypeIcon":
{"Path":"https://windchill.ptc.com/Windchill/wtcore/images/part.gif","Tooltip":"Part"},
"Identity":"0000019104, BatchPart2, Demo Organization, A","GeneralStatus":null,
"ShareStatus":null,"ChangeStatus":null,"Supersedes":null,
"AssemblyMode":{"Value":"separable","Display":"Separable"},
"DefaultUnit":{"Value":"ea","Display":"each"},
"DefaultTraceCode":{"Value":"0","Display":"Untraced"},
"Source":{"Value":"make","Display":"Make"},"ConfigurableModule":
{"Value":"standard","Display":"No"},"GatheringPart":false,
"PhantomManufacturingPart":false,"BOMType":null,"AlternateNumber":null,
"View":"","CheckoutState":"Checked in","CheckOutStatus":"",
"Comments":null,"State":{"Value":"INWORK","Display":"In Work"},
"LifeCycleTemplateName":"Basic","VersionID":"VR:wt.part.WTPart:601175",
"Revision":"A","Version":"A.0","Latest":true,"FolderName":null,
"CabinetName":"Default","FolderLocation":"/Default",
"OrganizationName":"Demo Organization",
"CreatedOn":"2010-04-20T08:32:51Z","LastModified":"2010-04-20T08:32:51Z"}
--changeset_2f39c1fb-0e4b-4a60-9e2d-89cab451db7f--

```
--batch_3487ef0a-9598-41cb-a3cf-1d8616ed9e58--
```

Batch Request with References Between Entity and Property Value

In the following example, there are five changesets with following content ID:

- `Content-ID: 1_1`—Creates a part with the name `PARENT`.
- `Content-ID: 2_1`—Creates a part with the name `CHILD`.
- `Content-ID: 3_1`—References part `PARENT` from `Content-ID: 1_1` and checks out the part.
- `Content-ID: 4_1`—References the checked out part `PARENT` from `Content-ID: 3_1`. Creates bind relationship with `CHILD`, which is referenced from `Content-ID: 2_1`.
- `Content-ID: 5_1`—References the checked out part `PARENT` from `Content-ID: 3_1` and checks in the part.

```
--batch_7b16726a-cc8b-4b51-a14d-9941f32c6859
Content-Type: multipart/mixed;boundary=
changeset_d0a2cc08-c5d6-466b-be37-26770b74716f


--changeset_d0a2cc08-c5d6-466b-be37-26770b74716f
Content-Id: 1_1
Content-Transfer-Encoding: binary
Content-Type: application/http

POST /Windchill/servlet/odata/v2/ProdMgmt/Parts HTTP/1.1
Accept: application/json
Content-Type: application/json

{"Context@odata.bind":"Containers('OR:wt.pdmlink.PDMLinkProduct:71864')",
"AssemblyMode":{"Value":"separable"},"DefaultUnit":{"Value":"ea"},
"GatheringPart":false,"DefaultTraceCode":{"Value":"0"},
"PhantomManufacturingPart":false,"Source":{"Value":"make"},
"Name":"PARENT"}

--changeset_d0a2cc08-c5d6-466b-be37-26770b74716f--

--batch_7b16726a-cc8b-4b51-a14d-9941f32c6859
Content-Type: multipart/mixed;boundary=changeset_
b9661f85-ff44-4b91-a7e3-1fed7bb27b72


--changeset_b9661f85-ff44-4b91-a7e3-1fed7bb27b72
Content-Id: 2_1
Content-Transfer-Encoding: binary
Content-Type: application/http

POST /Windchill/servlet/odata/v2/ProdMgmt/Parts HTTP/1.1
Accept: application/json
```

```
Content-Type: application/json

{"Context@odata.bind":"Containers('OR:wt.pdmlink.PDMLinkProduct:71864')",
"AssemblyMode":{"Value":"separable"},"DefaultUnit":{"Value":"ea"},
"GatheringPart":false,"DefaultTraceCode":{"Value":"0"},
"PhantomManufacturingPart":false, "Source":{"Value":"make"},
"Name":"CHILD"}

--changeset_b9661f85-ff44-4b91-a7e3-1fed7bb27b72--

--batch_7b16726a-cc8b-4b51-a14d-9941f32c6859
Content-Type: multipart/mixed;boundary=changeset_
8ea2b69a-f111-416c-96e4-08bde78e4008


--changeset_8ea2b69a-f111-416c-96e4-08bde78e4008
Content-Id: 3_1
Content-Transfer-Encoding: binary
Content-Type: application/http

POST $1_1/PTC.CheckOut HTTP/1.1
Accept: application/json
Content-Type: application/json

{}

--changeset_8ea2b69a-f111-416c-96e4-08bde78e4008--

--batch_7b16726a-cc8b-4b51-a14d-9941f32c6859
Content-Type: multipart/mixed;boundary=
changeset_5408e739-b840-47e6-8f31-bfcb8ffd63fd


--changeset_5408e739-b840-47e6-8f31-bfcb8ffd63fd
Content-Id: 4_1
Content-Transfer-Encoding: binary
Content-Type: application/http

POST $3_1/Uses HTTP/1.1
Accept: application/json
Content-Type: application/json

{"TraceCode":{"Value":"L"},"FindNumber":"F001","Quantity":2,
"Uses@odata.bind":"$2_1","Unit":{"Value":"ea"},"LineNumber":100}

--changeset_5408e739-b840-47e6-8f31-bfcb8ffd63fd--

--batch_7b16726a-cc8b-4b51-a14d-9941f32c6859
Content-Type: multipart/mixed;boundary=
changeset_d275447b-39b8-4e61-a5ca-cbed8f42739f
```

```
--changeset_d275447b-39b8-4e61-a5ca-cbed8f42739f
Content-Id: 5_1
Content-Transfer-Encoding: binary
Content-Type: application/http

POST $3_1/PTC.CheckIn HTTP/1.1
Accept: application/json
Content-Type: application/json

{}

--changeset_d275447b-39b8-4e61-a5ca-cbed8f42739f--

--batch_7b16726a-cc8b-4b51-a14d-9941f32c6859--
```

Getting Information About Windchill Constraints

In Windchill, you can apply constraints to attributes. The constraints are applied in the **Type and Attribute Manager** utility, or in **Object Initialization Rules**. In Windchill, the constraints are applied at site, organization, product, and library levels. For example, in the **Type and Attribute Manager** utility, you can apply a Legal value list constraint to an attribute. When you apply this constraint, the attribute can only take permitted values specified in the constraint.

Constraints can be applied using an OIR. For example, when an attribute like Number takes the value which is automatically generated by the server either at time of persistence or a pre-generated value before persistence.

Entity properties that map to Windchill type attributes adhere to the constraints that are applied by the server. To provide valid values for these properties, the clients must know the constraints that were applied in Windchill to these entity properties.

Windchill REST Services provides functions that can be used to query for constraints that have been applied to the entity properties. These functions are bound to containers and can be called for site, organization, product, or library containers. The following functions are available:

- `GetDriverProperties`—Gets the driver properties defined for the given entity type. Driver properties are used by the server to select the constraints applied to certain property values. For example, the following GET request calls the function to get the driver properties for a Part entity type:

  ```
  GET DataAdmin/Containers('OR:wt.pdmlink.PDMLinkProduct:99999')/
  ```

```
PTC.DataAdmin.GetDriverProperties(entityName='PTC.ProdMgmt.Part')
```

- `GetConstraints`—Gets the constraints defined on a specific entity property or on all properties of an entity type. For example, the following GET request calls the function to get the constraints on all the properties for a Part entity type:

```
GET DataAdmin/Containers('OR:wt.pdmlink.PDMLinkProduct:99999')/
PTC.DataAdmin.GetConstraints(entityName='PTC.ProdMgmt.Part',driverProperties=@props)
?props={"Items":[{"Name":"EndItem","Type":"Edm.Boolean","Value":"true"}]}
```

- `GetPregeneratedValue`—Gets the server generated value for an entity property for which a pre-generated constraint is set in the OIR. For example, the following GET request calls the function to get the pre-generated value for the property Number on entity type Part:

```
GET DataAdmin/Containers('OR:wt.pdmlink.PDMLinkProduct:99999')
/PTC.DataAdmin.GetPregeneratedValue(entityName='PTC.ProdMgmt.Part',
propertyName='Number',driverProperties=@props)
?props={"Items":[{"Name":"EndItem","Type":"Edm.Boolean","Value":"true"}]}
```

Getting Information About Windchill Life Cycle States

Business information and objects become more mature throughout the product development cycle. As the object progresses in the cycle, it moves through various life cycle states of maturity. In Windchill, every object type can have a unique set of life cycle states.

The life cycle administrator in Windchill can create various life cycle states for an object, and can associate these states with a template. A business object which has a life cycle template associated with it can transition between the defined life cycle states.

The function `GetValidStateTransitions()` returns the life cycle states that the entity can transition from its current state. The life cycle states are retrieved as `EnumType`.

The function is bound to an entity that has a life cycle template associated with it. The function is available for all the entities that are life cycle managed. The list of life cycle states returned by the function depends on the life cycle template which is associated to the bound entity.

If the URL used to execute the function is not formed correctly, the function throws the URL malformed exception.

Set the Life Cycle State of an Entity

An entity can transition through different life cycle states. If an entity is associated with a life cycle template, then the transition states depend on the unique set of life cycle states defined in the template.

Use the action `SetState()` to set a valid life cycle state for an entity. The action accepts the life cycle state as input parameter in the display-value pair format. The valid value for the life cycle state is of type `PTC.EnumType`.

For example, if you want to change the life cycle state of an entity to `INWORK`, pass the input parameter as:

```
{ "Display": "In Work" "Value": "INWORK" }
```

You must specify the value for both the parameters. They cannot be blank or null. You can specify any value for the `Display` parameter. In the `Value` parameter, specify the internal name of life cycle state. When the action succeeds, the life cycle state of the entity is changed to the value specified in the parameter `Value`.

If an invalid life cycle state is specified, then the action returns a bad request error. Use the function `GetValidStateTransitions()`, to get the list of valid life cycle states that the entity can transition. Refer to the section Getting Information About Windchill Life Cycle States on page 71, for more information.

If the URL used to execute the action is not formed correctly, the action throws the URL malformed exception.

Function to Get the Value of Nonce Token

Nonce is server generated token which helps in preventing cross-site request forgery (CSRF) attacks. REST clients must provide the token while creating, updating, or deleting the entities in the system.

In the framework, use the function `GetCSRFToken()` to get the value of the nonce token. The function is available in the PTC Common domain. To get the nonce value, use the URL:

```
https://<Windchill server>/Windchill/servlet/odata/PTC/GetCSRFToken()
```

The token is returned in the JSON response. For example the response is as shown below:

```
{
 "@odata.context": "https://windchill.ptc.com/Windchill/servlet/odata/v1/PTC/
$metadata#CSRFToken",
 "NonceKey": "CSRF_NONCE",
 "NonceValue": "8q87WtSxvWkSH9FMtsQUboOI5TtCS7gWh8RUb4OG ="
}
```

4

Windchill REST Services Domain Capabilities

PTC Domains

This section explains the domains provided by PTC in Windchill REST Services.

When you install Windchill REST Services, some domains defined by PTC are also installed. These domains enable you to work with Windchill types in the REST architecture. You can also create new custom domains, or extend an existing domain to enable more entities.

A domain in Windchill REST Services represents a RESTful web service, which follows the OData standard. A domain describes its Entity Data Model (EDM) by defining the entity sets, relationships, entity types, and operations.

Overview

This section explains the domains provided by PTC in Windchill REST Services.

The following domains are provided as a part of Windchill REST Services:

- *ProdMgmt*—**PTC Product Management** domain exposes entities representing parts and BOMs, Windchill objects that are most frequently used while developing products. See the section PTC Product Management Domain on page 76, for more information on the domain.
- *DocMgmt*—**PTC Document Management** domain provides entities that enable users to manage Windchill documents (`WTDocuments`). See the section PTC Document Management Domain on page 85, for more information on the domain.
- *DataAdmin*—**PTC Data Administration** domain provides entities that enable users to manage data containers such as, organizations, products, libraries and projects in Windchill. See the section PTC Data Administration Domain on page 86, for more information on the domain.
- *PrincipalMgmt*—**PTC Principal Management** domain provides entities that work with Windchill groups and users. See the section PTC Principal Management Domain on page 92, for more information on the domain.
- *PTC*—**PTC Common** domain provides some utility entity types that are commonly used. See the section PTC Common Domain on page 93, for more information on the domain.
- *NavCriteria*—**PTC Navigation Criteria** domain provides entities that access the filters available in a part structure in Windchill. See the section PTC Navigation Criteria Domain on page 98, for more information on the domain.
- *DynamicDocMgmt*—**PTC Dynamic Document Management** domain provides entities that work with dynamic documents of Windchill. See the section PTC Dynamic Document Management Domain on page 102, for more information on the domain.

- *PartListMgmt*—**PTC Parts List Management** domain provides entities that work with parts list and parts list items of Windchill. See the section PTC Parts List Management Domain on page 104, for more information on the domain.
- *ServiceInfoMgmt*—**PTC Service Information Management** domain provides entities that work with objects and structures of Windchill Service Information Manager. See the section PTC Service Information Management Domain on page 105, for more information on the domain.
- Quality domains—The domains provide entities that work with Quality Management Services of Windchill. The Quality domains are available only if you have installed the relevant Quality products during Windchill installation. See the section PTC Quality Domains on page 109, for more information on the domain.

  The following Quality domains are available:

  - PTC Quality Management System Domain on page 109
  - PTC Nonconformance Domain on page 111
  - PTC Customer Experience Management Domain on page 112
  - PTC Regulatory Master Domain on page 113
  - PTC CAPA Domain on page 114
  - PTC Audit Domain on page 115
- *IE*—**PTC Info*Engine System** domain provides entities that work with Info*Engine tasks of Windchill. See the section PTC Info*Engine System Domain on page 116, for more information on the domain.
- *Factory*—**PTC Factory** domain provides entities that work with the manufacturing data management capabilities of Windchill. The domain is available only if you have installed Windchill MPMLink. See the section PTC Factory Domain on page 117, for more information on the domain.
- *MfgProcMgmt*—**PTC Manufacturing Process Management** domain provides entities that work with the manufacturing process management capabilities (MPM) of Windchill. The domain is available only if you have installed Windchill MPMLink. See the section PTC Manufacturing Process Management Domain on page 119, for more information on the domain.
- *ChangeMgmt*—**PTC Change Management** domain provides entities that work with the change management capabilities of Windchill. See the section PTC Change Management Domain on page 123, for more information on the domain.
- *ClfStructure*—**PTC Classification Structure** domain provides access to the classification structure and classification nodes in Windchill. See the section PTC Classification Structure Domain on page 126, for more information on the domain.

- *SavedSearch*—**PTC Saved Search** domain provides access to saved searches in Windchill. See the section PTC Saved Search Domain on page 130, for more information on the domain.
- *Visualization*—**PTC Visualization** domain provides access to visualization services of Windchill. See the section PTC Visualization Domain on page 132, for more information on the domain.
- *ProdPlatformMgmt*—**PTC Product Platform Management** domain provides access to Options and Variants capabilities of Windchill. See the section PTC Product Platform Management Domain on page 135, for more information on the domain.
- *CADDocumentMgmt*—**PTC CAD Document Management** domain provides access to CAD data management capabilities of Windchill. See the section PTC CAD Document Management Domain on page 138, for more information on the domain.
- *EventMgmt*—**PTC Event Management** domain provides access to the webhook subscription capabilities of Windchill. See the section PTC Event Management Domain on page 145, for more information on the domain.
- *SupplierMgmt*—**PTC Supplier Management** domain provides access to the supplier management capabilities of Windchill. See the section PTC Supplier Management Domain on page 148, for more information on the domain.
- *Workflow*—**PTC Workflow** domain provides access to the workflow capabilities of Windchill. See the section PTC Workflow Domain on page 149, for more information on the domain.

PTC Product Management Domain

The PTC Product Management domain provides access to the product management capabilities of Windchill. It provides OData entities that represent business objects, such as, Part and BOM. The following table shows the Windchill items that are enabled with OData entities in the Product Management domain. The Product Management domain references the PTC Document Management domain to provide navigations to reference and describe documents.

You can work with classified parts in the PTC Product Management domain only if Windchill PartsLink module is installed.

You can create classified parts. You can also update the classification attributes of an existing classified part. When classifying a part, if you specify the incorrect classification node name, or incorrect classification attribute name and value, relevant error messages are returned.

The supplier management entities such as, `SupplierPart`, `ManufacturerPart`, `VendorPart`, `AXLEntry`, and so on are available in PTC Product Management domain. These entities are available only if Supplier Management module is installed in Windchill. Classification is supported for

manufacturer and vendor parts. See the section PTC Supplier Management Domain on page 148, for more information about PTC Supplier Management domain.

The following table lists the significant OData entities available in the Product Management domain. To see all the OData entities available in the Product Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Part	`Part` `ElectricalPart`	The `Part` entity represents a part version. In Windchill, use the `WTPart` and `WTPartMaster` classes to work with part versions. `ElectricalPart` is derived from `Part` and represents the soft type that is available in Windchill. This entity provides navigation property `AssignedOptionSet` that retrieves the option set assigned to the part. You can also expand the navigation property to get detailed information about the option set. See the section PTC Product Platform Management Domain on page 135, for more information about `OptionSet` entity. **Note** The navigation property is available only if the option `Configurable Module Support` is set to *Yes* in **Utilities ► Preference Management ► Options and Variants**.

Bill of material	`BOM`
	`PartUse`
	`UsageOccurrence`
		The `BOM` entity represents the part structure expanded to some levels.

Items	OData Entities	Description
		`PartUse` is an OData entity that represents the association between parent and child parts. It has attributes such as, quantity, unit, line number, and so on. These attributes of entity models are also available in the `WTPartUsageLink` class. The `UsageOccurrence` entity represents the reference designator when a component is used multiple times in a BOM.
Part that resides in a Windchill folder	`PartContent`	This entity is derived from `FolderContent` entity that is available in the `DataAdmin` domain. The entity represents a part residing in a folder.
Supplier part	`SupplierPart`	Supplier part is a subtype of part. You can perform only read operation on this entity.
Manufacturer part	`ManufacturerPart`	Manufacturer part is a subtype of supplier part. It is produced by a manufacturer other than the Original Equipment Manufacturer (OEM). The `AXLEntry` entity is used to associate the `ManufacturerPart` entity with the `Part` entity.

Items	OData Entities	Description
Vendor part	`VendorPart`	Vendor part is a subtype of supplier part. It is a part that is supplied by the vendor. The `AXLEntry` entity is used to associate the `VendorPart` entity with the `Part` entity.
AXLEntry (AML/AVL)	`AXLEntry`	`AXLEntry` entity represent the association between `SupplierPart`, that is, `ManufacturerPart` and `VendorPart` and the `Part` entity. An OEM part can be associated with several manufacturer parts or vendor parts. The `AXLEntry` entity is available as navigation property on the `Part` entity, which retrieves the manufacturer and vendor parts that are associated with the part.

Organization Navigation Property

Use the `Organization` navigation property to

Navigation URLs for AssignedOptionSet

You can use the following URLs to retrieve information about assigned option set that is associated with product or library containers.

- To get all the parts with their assigned option sets:

  `/ProdMgmt/Parts?$expand=AssignedOptionSet`
- To get a specific part along with its assigned option set:

  `/ProdMgmt/Parts(<oid>)?$expand=AssignedOptionSet`

Actions Available in the PTC Product Management Domain

The following actions are available in the PTC Product Management domain:

GetBOM

The action `GetBOM` returns the bill of materials (BOM) for the product structure. The action is bound to the entity `NavigationCriteria`, that is, to the filter saved in Windchill.

When you call the `GetBOM` action, in the request body of the URL, you can specify the ID of the `NavigationCriteria`. This is the ID of the saved filter you want to use as the filter criteria. If you do not specify the ID of the filter in the request body, then the default filter is used to work with the product structure. Alternatively, you can specify the navigation criteria in the request payload.

This action will be deprecated in a future release of Windchill REST Services. Use the action `GetPartStructure` instead.

GetBOMWithInlineNavCriteria

The action `GetBOMWithInlineNavCriteria` returns the bill of materials (BOM) for the product structure. Pass the navigation criteria as the input parameter. The action is bound to the entity `NavigationCriteria`, that is, to the filter saved in Windchill. You can pass partial representation of the navigation criteria.

If the navigation criteria contains the property `ApplyToTopLevelObject`, which is set as `True`, and no qualifying version is found for the top level object, a relevant error message is returned.

For example, the following partial navigation criteria returns the Bill of Material for Manufacturing view that includes working copies of parts.

```
POST /ProdMgmt/Parts(<oid>)/PTC.ProdMgmt.GetBOMWithInlineNavCriteria
{
  "NavigationCriteria": {
      "ApplyToTopLevelObject": true,
      "ConfigSpecs": [
        {
          "@odata.type": "#PTC.NavCriteria.WTPartStandardConfigSpec",
          "WorkingIncluded": true,
          "View": "Manufacturing"
        }
      ]
  }
}
```

This action will be deprecated in a future release of Windchill REST Services. Use the action `GetPartStructure` instead.

GetPartStructure

The action `GetPartStructure` returns the bill of materials (BOM) for a product structure along with path details for occurrences. The action is bound to the entity `NavigationCriteria`, that is, to the filter saved in Windchill.

When you call the `GetPartStructure` action, in the request body of the URL, you can specify the ID of the `NavigationCriteria`. This is the ID of the saved filter you want to use as the filter criteria. If you do not specify the ID of the filter in the request body, then the default filter is used to work with the product structure. Alternatively, you can specify the navigation criteria in the request payload.

As compared to the `GetBOM` and `GetBOMWithInlineNavCriteria` actions, when you call the `GetPartStructure` action, the following additional URLs are returned:

- `PathId` is the occurrence path of the component part in the BOM structure. The complete path from the root of the BOM structure is returned. This URL can be used in path filters to filter on the specific component.
- `PVTreeId` is the occurrence path of the component part in the viewable file. The complete path from the root of the BOM structure is returned. This URL can be used to work with Visualization tree. For example, in an application you consume this URL and highlight the component in the Visualization tree.
- `PVParentTreeId` is the occurrence path to the parent of the component part in the viewable file. The complete path from the root of the BOM structure is returned.

When you call the `GetPartStructure` action along with expand on occurrences, the component along with its details is returned as many times as the component is available in the BOM structure.

For example, consider a part A1 which has the following components:

- Component C1—Quantity 2 with occurrences R1 and R2
- Component C2—Quantity 1 with no occurrences

When you use the `GetPartStructure` action with occurrences to get the BOM, the following response is returned:

```
{
  "PartName": "A1"
  "PartUseId": null,
  "Part": {
      "ID": "<oid>",
      "Name": "A1",
      ...
  },
  "PartUse": null,
  "Ocurrences": [],
  "Components": [
      {
        "PartName": "C1"
        "PartUseId": "<linkoid>",
        "PartId": "<pathidofcomponent>",
```

```
  "PVTreeId": "<treeidfromviz>",
  "PVParentTreeId": "<parenttreeidfromviz>",
  "Part": {
      "ID": "<oid>",
      "Name": "C1",
      ...
  },
  "PartUse": {
      "ID": "<linkoid>",
      "Qty": 1,
      ...
  },
  "Ocurrence": {
        "ID": "<occoid>",
        "ReferenceDesignator": "R1"
        ...
  },
  "Components": []
},
{
  "PartName": "C1"
  "PartUseId": "<linkoid>",
  "PartId": "<pathidofcomponent>",
  "PVTreeId": "<treeidfromviz>",
  "PVParentTreeId": "<parenttreeidfromviz>",
  "Part": {
      "ID": "<oid>",
      "Name": "C1",
      ...
  },
  "PartUse": {
      "ID": "<linkoid>",
      "Qty": 1,
      ...
  },
  "Ocurrence": {
        "ID": "<occoid>",
        "ReferenceDesignator": "R2"
        ...
  },
  "Components": []
},
{
  "PartName": "C2"
  "PartUseId": <linkoid>",
  "PartId": "<pathidofcomponent>",
```

```
        "PVTreeId": "<treeidfromviz>",
        "PVParentTreeId": "<parenttreeidfromviz>",
        "Part": {
            "ID": "<oid>",
            "Name": "C2",
            ...
        },
        "PartUse": {
            "ID": "<linkoid>",
            "Qty": 1,
            ...
        },
        "Ocurrences": [],
        "Components": []
      }
  ]
}
```

GetPartsList

The action `GetPartsList` returns a consolidated flat list of components for the specified part structure. The action returns:

- Number of occurrences, that is, quantity of the component in the part list. Each component is only listed once with a consolidated total quantity.
- Unit of the component

To get more details on the components of a part structure, specify the value `application/json;odata.metadata=full` in the `Accept` header of the HTTP request. The `odata.metadata` parameter controls how much information is included in the response. When you specify the value as full, the response includes all the information.

When you specify the `OID` of the part and `Accept` header value `application/json;odata.metadata=full` in the URL request, the action `GetPartsList()` returns the following information:

- OData type for the part component
- Quantity of the part component in the specified part structure
- Unit of the component
- Navigation URL to the part component

You can expand the `Part` and `PartUses` navigation properties for more details of the components.

UpdateCommonProperties

The action `UpdateCommonProperties` edits the common properties of parts. The action is available only if you set the property *hasCommonProperties* to `true` in the `Parts.json` file.

The action must not be called on objects that are checked out.

Common part attributes, `Name`, `Number`, `EndItem`, `DefaultUnit`, `DefaultTraceCode`, `ConfigurableModule`, `GatheringPart`, `Organization`, and `PhantomManufacturingPart` can be edited.

Function Available in the PTC Product Management Domain

The following function is available in the PTC Product Management domain:

GetAssignedExpression()

The function `GetAssignedExpression()` returns the assigned expressions for `Part`, `PartUse`, and `UsageOccurrence` entities. The function is bound to these entities. You can specify a single entity as the input parameter. Basic and advanced types of expressions are supported.

The information related to assigned expressions is retrieved using the complex type, `AssignedExpression`, which is defined in the PTC Product Platform Management domain.

To retrieve the assigned expressions for multiple objects, use the `GetAssignedExpressions` action defined in the PTC Product Platform Management domain. See the section Action Available in the PTC Product Platform Management Domain on page 137, for more information.

Both dependent and independent expression modes are supported.

PTC Document Management Domain

The Document Management domain provides access to the document management capabilities of Windchill. It enables you to create documents. You can also upload and download content from documents.

The following table lists the significant OData entities available in the Document Management domain. To see all the OData entities available in the Document Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Business document	`Document`	The `Document` entity represents a document version. In Windchill, use the `WTDocument` and `WTDocumentMaster` classes to work with document versions.
Content information	`ContentInfo`	The `ContentInfo` entity contains content information which is used in Stage 3 for uploading content to the document.

Action Available in the PTC Document Management Domain

The following action is available in the PTC Document Management domain:

UpdateCommonProperties

The action `UpdateCommonProperties` edits the common properties of documents. The action is available only if you set the property *hasCommonProperties* to `true` in the `Documents.json` file.

The action must not be called on objects that are checked out.

In this release, you can only edit the attributes `Name`, `Number`, and `Organization`.

PTC Data Administration Domain

The PTC Data Administration domain provides access to data administration capabilities of Windchill. The domain includes entities that represent Windchill containers such as, site, organization, product, libraries, project containers, and so on. It also includes entities that represent the folder hierarchy in these containers. This domain contains an entity set called `Containers` that enables clients to read the containers available in their Windchill system.

Note

`Containers` entity set is read-only, and does not support update, delete and create operations.

The following table lists the significant OData entities available in the Data Administration domain. To see all the OData entities available in the Data Administration domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Windchill container	`Container`	A Windchill container. This entity exposes only those attributes that are common across all types of containers.
Site container	`Site`	A site container which is derived from the `Container` entity.
Organization container	`OrganizationContainer`	An organization container, which is derived from the `Container` entity.
Product container	`ProductContainer`	A product container, which is derived from the `Container` entity. This entity provides the following navigation properties: • `OptionPool` — Retrieves the options group and options available in the option pool for the specified container. See the section PTC Product Platform Management Domain on page 135, for more information about `OptionPoolItem` entity. • `AssignedOption-Set`—Retrieves the option set assigned to the specified container. See the section PTC Product Platform Management Domain on page 135,

Items	OData Entities	Description
		for more information about `OptionSet` entity. • `OptionPoolA-liases`—Retrieves the expression aliases assigned to the specified container. **Note** These navigation properties are available only if the option `Configurable Module Support` is set to *Yes* in **Utilities ▸ Preference Management ▸ Options and Variants**.
Library container	`LibraryContainer`	A library container This entity provides the following navigation properties: • `OptionPool` — Retrieves the options group and options available in the option pool for the specified container. See the section PTC Product Platform Management Domain on page 135, for more information about `OptionPoolItem` entity. • `AssignedOption-Set`—Retrieves the option set assigned to the specified container. See the

Items	OData Entities	Description
		section PTC Product Platform Management Domain on page 135, for more information about `OptionSet` entity. • `OptionPoolA-liases`—Retrieves the expression aliases assigned to the specified container. **Note** These navigation properties are available only if the option `Configurable Module Support` is set to *Yes* in **Utilities ► Preference Management ► Options and Variants**.
Project container	`ProjectContainer`	A project container

Items	OData Entities	Description
Folders and subfolders	`Folder`	A folder, which is derived from the `Container` entity. You can use folders to organize objects. You can create, update, and delete folders and subfolders.
Generic item that resides in the Windchill folder	`FolderContent`	The `FolderContent` entity represents the generic view of an item that resides in a folder. Other domain entities can derive from this entity to create more specific views. For example, in the Product Management domain, the `PartContent` entities derive from `FolderContent`.

Navigation URLs for OptionPool, AssignedOptionSet, and OptionPoolAliases

You can use the following URLs to retrieve option pool items (option groups and options), an assigned option set, and expression aliases. These navigation properties are available for product and library containers.

- `OptionPool` Navigation Property:
  - To get the option pool items for a specific container:
    - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer(<oid>)/OptionPool`
    - `/DataAdmin/Containers/PTC.DataAdmin.LibraryContainer(<oid>)/OptionPool`
  - To get the containers along with the option pool items:
    - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer?$expand=OptionPool`

        - `/DataAdmin/Containers/PTC.DataAdmin.LibraryContainer?$expand=OptionPool`
    - To get the option groups from the option pool for a specific container typecast to the `OptionGroup` entity:
        - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer(<oid>)/OptionPool/PTC.ProdPlatformMgmt.OptionGroup`
        - `/DataAdmin/Containers/PTC.DataAdmin.LibraryContainer(<oid>)/OptionPool/PTC.ProdPlatformMgmt.OptionGroup`
    - To get the top-level options from the option pool for a specific container typecast to the `Option` entity:
        - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer(<oid>)/OptionPool/PTC.ProdPlatformMgmt.Option`
        - `/DataAdmin/Containers/PTC.DataAdmin.LibraryContainer(<oid>)/OptionPool/PTC.ProdPlatformMgmt.Option`
- `AssignedOptionSet` Navigation Property:
    - To get the assigned option set with details:
        - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer?$expand=AssignedOptionSet`
        - `/DataAdmin/Containers/PTC.DataAdmin.LibraryContainer?$expand=AssignedOptionSet`
    - To get the assigned option set with details for a specific container:
        - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer(<oid>)?$expand=AssignedOptionSet`
        - `/DataAdmin/Containers/PTC.DataAdmin.LibraryContainer(<oid>)?$expand=AssignedOptionSet`
- `OptionPoolAliases` Navigation Property:
    - To get the expression aliases with details:
        - `/DataAdmin/Containers/PTC.DataAdmin.ProductContainer?$expand=OptionPoolAliases`

- /DataAdmin/Containers/PTC.DataAdmin.LibraryContainer?$expand=OptionPoolAliases

- To get the expression aliases with details for a specific container:
  - /DataAdmin/Containers/PTC.DataAdmin.ProductContainer(<oid>)?$expand=OptionPoolAliases
  - /DataAdmin/Containers/PTC.DataAdmin.LibraryContainer(<oid>)?$expand=OptionPoolAliases

PTC Principal Management Domain

The Principal Management domain provides read access to the information related to principals in Windchill. The Windchill principals can be users, groups, license groups and so on.

The following table lists the significant OData entities available in the Principal Management domain. To see all the OData entities available in the Principal Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Windchill principal	`Principal`	The `Principal` entity represents the generic view of a Windchill principal.
Windchill user	`User`	The `User` entity represents a principal who is the user. In Windchill, use the `WTUser` class to work with users.
Windchill group	`Group`	The `Group` entity represents a principal who is the group. In Windchill, use the `WTGroup` class to work with groups.

Items	OData Entities	Description
Windchill organization principal	`Organization`	The `Organization` entity represents a Windchill group that is an organization principal. In Windchill, use the `WTOrganization` class to work with organization principals.
License group	`LicenseGroup`	The `LicenseGroup` entity represents a Windchill license groups. Windchill license groups are available at the site level to help you manage your license compliances. You must add the user to the appropriate license group depending on the licenses allocated to them. This entity provides navigation property that lists all the license groups for a user. You can also expand the navigation property. License groups are access controlled.

PTC Common Domain

PTC Common (PTC) domain provides access to entities that are common to multiple domains. It is recommended to store common entities in this domain. The domain also provides complex types and functions that are used in other domains.

The following table lists the significant OData entities available in the PTC Common domain. To see all the OData entities available in the PTC Common domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Content file associated to a business document	`ContentItem`	The `ContentItem` entity provides a generic view of the content that is associated to a business document. More specialized entities are derived from `ContentItem` are `ExternalStoredData`, `ApplicationData`, and `URLData`. You can use PATCH requests to update the `Comments` and `Description` properties. In the body of the PATCH request, specify the values for the attributes.
Content stored in an external location	`ExternalStoredData`	The `ExternalStoredData` entity provides a specialized view of an externally stored `ContentItem`, which is stored in a document
A URL that is stored in a business document	`URLData`	The `URLData` entity provides a specialized view of the URL `ContentItem` that is stored in a document

Items	OData Entities	Description
Content stored in Windchill application	`ApplicationData`	The `ApplicationData` entity provides a specialized view of the content stored by the Windchill application.
Windchill Objects	`WindchillEntity`	Some of the Windchill objects types are not available in Windchill REST Services. All the object types that are not available in Windchill REST Services are represented as `WindchillEntity`. `WindchillEntity` returns only the ID, created by, and last modified by, attributes for these objects. If you want to return Windchill objects that are not available in Windchill REST Services, use this entity to represent such objects. The object types, which are available in Windchill REST Services, are automatically mapped to the relevant entity type.

In addition to the entities, this domain also contains the following complex types which represent:

- `QuantityOfMeasureType`—Real number with unit data type in Windchill.
- `Hyperlink`—URL data type in Windchill.
- `Icon`—Icon in Windchill.
- `EnumType`—Attributes that are enumerated types in Windchill or attributes that have type constraints defined.

- `ClassificationInfo`—Classification binding attribute.
- `ClassificationAttribute`—Classifications attributes.
- `EntityMetaInfo`—Windchill metadata information for entity types.
- `PropertyMetaInfo`—Windchill metadata information for properties.

Functions Available in the PTC Common Domain

The following functions are available in the PTC Common domain:

GetEnumTypeConstraint()

The function `GetEnumTypeConstraint()` is used to query the valid values for a property, which are represented as `EnumType`. These values are used for implementing validations on client side.

GetAllStates()

The function `GetAllStates()` returns a list of life cycle states, which are available and can be selected in Windchill. The life cycle states that cannot be selected in Windchill are not returned. The life cycle states are retrieved from the `StateRb.rbinfo` file.

`GetAllStates()` is an unbound function, which is available in all the domains which import the PTC Common domain.

If the URL used to execute the function is not formed correctly, the function throws the URL malformed exception.

GetWindchillMetaInfo()

The `GetWindchillMetaInfo()` function returns the Windchill metadata for OData entity types and properties that are available in the domain from which the function is called. The function is available in the all domains that import the PTC Common domain.

The function uses the complex types `EntityMetaInfo` and `PropertyMetaInfo` to retrieve the Windchill metadata.

Note the following points while retrieving the Windchill metadata using the `GetWindchillMetaInfo()` function:

- For entity types that are backed by Windchill types, the display and internal names of the entity are retrieved.
- For entity types that are not backed by Windchill types, the display and internal names of the entity are retrieved as `null`.
- For properties that are available on the OData entity type, but are not available in Windchill, the value of display name is retrieved as `null`.

For example, consider the `VersionID` property which is available on many OData entity types. For this property, there is no equivalent property in Windchill. In this case, the display name is always retrieved as `null`.

- The function returns internal names and localized display names for Windchill types and properties depending on the language that you have set.

For example, if you call the function from the PTC Product Management domain, the following response is returned.

Request URL

`GET ProdMgmt/GetWindchillMetaInfo()`

The response is:


```
[
  {
    "EntityType": "PTC.ProdMgmt.Part",
    "BaseType": "PTC.WindchillEntity",
    "HasWindchillType": true,
    "DisplayName": "Part",
    "InternalName": "wt.part.WTPart",
    "PropertyInfo": [
      {
         "PropertyName": "DefaultUnit",
         "DisplayName": "Default Unit",
         "InternalName": "defaultUnit"
       },
       {
         "PropertyName": "EndItem",
         "DisplayName": "End Item",
         "InternalName": "endItem"
       },
         ...
       ]
  },
  {
         "EntityType": "PTC.ProdMgmt.PartUse",
         "BaseType": "PTC.WindchillEntity",
         "HasWindchillType": true,
         "DisplayName": "Part Usage Link",
         "InternalName": "wt.part.WTPartUsageLink",
         "PropertyInfo": [
       {
         "PropertyName": "QuantityUnit",
         "DisplayName": "Quantity Unit",
         "InternalName": "quantityUnit"
       },
       {
         "PropertyName": "LineNumber",
         "DisplayName": "Line Number",
         "InternalName": "lineNumber"
       },
         ...
       ]
    },
    ...
```

```
]
```

To get the Windchill metadata for a specific entity in the domain from which the function is called, specify the following URL:

```
GET <Domain_Name>/GetWindchillMetaInfo(EntityName=
'<name_of_the_entity>')
```

For example, to get information about the `Part` entity, use the following URL:

```
GET ProdMgmt/GetWindchillMetaInfo(EntityName=
'PTC.ProdMgmt.Part')
```

PTC Navigation Criteria Domain

The PTC Navigation Criteria domain provides access to the filters available in a part structure in Windchill. The filters are divided into two categories, configuration specifications that display a complete bill of material, and specialized filters that show a subset of parts relevant to a design task. The following table shows the Windchill items that are enabled with OData entities in the PTC navigation Criteria domain.

The following table lists the significant OData entities available in the PTC Navigation Criteria domain. To see all the OData entities available in the PTC Navigation Criteria domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Saved filters or navigation criteria in Windchill	`NavigationCriteria`	The `NavigationCriteria` entity represents filters, which are created and saved in Windchill. The criteria defined in the filter is used when you work with product structures. To define the filter criteria, specify and save values in the filter. The values specified in the filter are access-controlled as is `NavigationCriteria`. The entity includes complex types that represent different types of configuration specifications, spatial, option, and attribute filters.

In addition to the entity, this domain also contains the following complex types:

- `ConfigSpec`—Collection of configuration specifications in a part and CAD document structure. A configuration specification filters the part or CAD document structure to show complete product configurations with a part and document version displayed for each structure node.

> **Note**
>
> If `ConfigSpec` is not specified for CAD documents, then no configuration specification is used, and all the children are unresolved. The document masters are returned for the CAD documents.

The configuration specification filters are represented as the following complex types:

- **Date Effectivity** configuration specification—View the configuration of a product based on a specified date, or date and time effectivity. This filter is available only for parts. Use the `WTPartEffectivityDateConfigSpec` complex type to work with parts.
- **As Matured** configuration specification—View the parts that are in their most mature state with reference to the specified date. This filter is available only for parts. Use the `WTPartAsMaturedConfigSpec` complex type to work with parts.
- **Unit Effectivity** configuration specification—View the configuration of a product based upon the serial number or lot number. This filter is available only for parts. Use the `WTPartEffectivityUnitConfigSpec` complex type to work with parts.
- **Baseline** configuration specification—Get the design from a specific event in time, that is, from a previously created baseline. Depending on the type of Windchill object use the following complex types:
  - `WTPartBaselineConfigSpec`—To work with parts.
  - `EPMDocBaselineConfigSpec`—To work with CAD documents.
- **Promotion Request** configuration specification—Get the data related to the specified promotion request. Depending on the type of Windchill object use the following complex types:
  - `WTPartPromotionNoticeConfigSpec`—To work with parts.
  - `EPMDocPromotionNoticeConfigSpec`—To work with CAD documents.
- **Latest** configuration specification—Find the latest designs, that is, the most recently created versions of a selected view and life cycle state. Depending on the type of Windchill object use the following complex types:
  - `WTPartStandardConfigSpec`—To work with parts.
  - `EPMDocStandardConfigSpec`—To work with CAD documents.

> **Note**
>
> If you specify the navigation criteria in the request body with only `ApplicableType` parameter set, the URL returns unresolved dependents for the CAD structure. However, if additionally, the `UseDefaultForUnresolved` parameter is also set to `true` in the request body, then **Latest** configuration specification is applied to the structure.

- **As Stored** configuration specification—Get the most recent configuration stored for a CAD document. This filter is available only for CAD documents. Use the `EPMDocAsStoredConfigSpec` complex type to work with CAD documents. If this filter is used with other Windchill object types, an error message is returned.

- `Filter`—Collection of specialized filters in a part structure. The specialized part structure filters reduce the complexity of the part structure by showing only those parts that are relevant to a design task or optional product configuration. The specialized filters are represented as the following complex types:
  - `AttributeFilter`—Attribute filters use part and usage link attribute information to determine what parts to include or exclude in the display of part structure.
  - `OptionFilter`—Option filter is used to include or exclude parts in a part structure based on expressions assigned to the parts.
  - `SpatialFilter`—Spatial filters use volumetric information to determine what components to display or hide in a part or CAD document structure. It is represented by following complex types:
    - `ProximitySpatialFilter`
    - `SphereSpatialFilter`
    - `BoxSpatialFilter`
  - `PathFilter`—Filters the display of large assemblies so that only the subassemblies that you are currently working on are displayed.
  - `OccurrencePathFilter`—Filters the display based on basic or advanced expressions assigned to an occurrence.
  - `UsagePathFilter`—Filters the display based on expressions assigned to usage links and parts.

If you have a spherical spatial filter with bounding box set as partial, the navigation criteria for this is represented as shown below in Windchill REST Services. Use the following URL to retrieve the navigation criteria:

```
GET /Windchill/servlet/odata/NavCriteria/NavigationCriterias?$filter=Name eq
'Sphere_filter001'
```

The response is as follows:

```
"value": [
        {
            "ID": "OR:wt.filter.NavigationCriteria:261602",
            "Name": "Sphere_filter001",
            "ApplyToTopLevelObject": true,
            "UseDefaultForUnresolved": false,
            "SharedToAll": false,
            "ApplicableType": "wt.part.WTPart",
            "Centricity": false,
            "HideUnresolvedDependents": false,
```

```
            "Filters": [
                {
                    "@odata.type": "#PTC.NavCriteria.SphereSpatialFilter",
                    "SpatialMethod": {
                        "Value": "PARTIALLY_IN",
                        "Display": "Partially in"
                    },
                    "XCenter": -0.5334005391706416,
                    "YCenter": 0.0503675,
                    "ZCenter": 0.00136,
                    "Radius": 0.40648001432418823
                    "Unit": "m"
                }
            ],
            "ConfigSpecs": [
                {
                    "@odata.type": "#PTC.NavCriteria.WTPartStandardConfigSpec",
                    "WorkingIncluded": true,
                    "View": "Design",
                    "LifeCycleState": null,
                    "Variation1": null,
                    "Variation2": null
                }
            ],
            "CreatedOn": "2018-10-01T11:38:16Z",
            "LastModified": "2018-10-01T11:38:16Z"
        }
    ]
}
```


PTC Dynamic Document Management Domain

PTC Dynamic Document Management domain provides access to the dynamic document capabilities of Windchill. Dynamic documents are compound documents that are encoded in a markup language such as XML. Dynamic documents include files that are authored in Arbortext Editor and can also contain other document-related files, such as graphics. Dynamic documents can contain parent and child documents, where the child documents can be transcluded into the parent. Dynamic documents can also reference other dynamic documents. With this domain, you can manage and share dynamic documents in Windchill.

Note

If you add `<image>` elements to a dynamic document, which is in DITA format, and check in the document, links to the graphics are automatically created in Windchill. If you delete the `<image>` elements, the links are retained.

In XML content, dynamic documents are identified by the `CADName` attribute of the document. OData endpoint can find dynamic documents with either the object identifier `Oid` or the `CADName` attribute.

The following table lists the significant OData entities available in the PTC Dynamic Document Management domain. To see all the OData entities available in the PTC Dynamic Document Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Dynamic document	`DynamicDocuments`	The `DynamicDocuments` entity represents a compound XML document that can be used to generate formatted output. In Windchill, dynamic documents are associated with `EPMMemberLinks` and `EPMReferenceLinks` classes. The entity provides two navigation properties to support Service Information Manager Translation. • Navigation from a dynamic document to the associated translated dynamic documents is possible. • For a Creo Illustrate 3D illustration file (`.c3di`), navigation is provided from the document to the associated XLIFF dynamic document. XLIFF contains translatable strings of the illustration.

PTC Parts List Management Domain

PTC Parts List Management (PartListMgmt) domain provides access to parts lists and parts list items, which are available in Windchill Service Parts module. Windchill Service Parts transforms eBOMs and mBOMs into service BOMs (sBOMs). A service BOM is a list of components in any order. Transforming the service BOM and the service BOM graphical representation into a parts list

enables you to order the components in a meaningful way. Use the PTC Parts List Management domain to retrieve part lists, part list items, illustrations, substitute, and supplementary parts.

The following table lists the significant OData entities available in the PTC Parts List Management domain. To see all the OData entities available in the PTC Parts List Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Parts list	`PartList`	A parts list is an ordered list of parts, which also includes metadata, such as, information about the conditions under which the parts are used.
Part list item	`PartListItem`	A part item list is a reference to a part in the parts list.
Illustrations	`Illustration`	An illustration is a graphic document that is linked to a parts list.

PTC Service Information Management Domain

PTC Service Information Management (ServiceInfoMgmt) domain provides access to the objects and structures of Windchill Service Information Manager.

Use Windchill Service Information Manager to organize service content, specify the content applicability rules and create publications. You can also automatically deliver the service information for every product configuration throughout the product lifecycle. PTC Service Information Management domain is available only if you install Windchill Service Information Manager.

Note

You should not call entities of other domains which are imported in PTC Service Information Management domain. For example, PTC Service Information Management domain imports PTC PartList Management domain. It is recommended not to call PartListMgmt entities such as, `PartList`, `PartListItem`, `Supplement`, and so on from the ServiceInfoMgmt domain.

The following table lists the significant OData entities available in the PTC Service Information Management domain. To see all the OData entities available in the PTC Service Information Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Information structure	`InformationStructure`	A hierarchical structure that organizes information related to a product.
Information group	`InformationGroup`	In an information structure, subgroups of information are referred to as information groups. These groups organize content.
Publication structure	`PublicationStructure`	Organizes service information associated with an information structure. Or, organizes service information that is stored independently in Windchill into a hierarchical structure that can be published.
Publication section	`PublicationSection`	A section of content in the publication structure.
Textual information element	`TextualInformationElement`	A textual content object in an information structure or publication structure. A textual information element is related to a dynamic document in the Dynamic Document Management domain. The dynamic document must contain XML data.
Graphical information element	`GraphicalInformationElement`	A graphical content object in an information structure or publication structure.

Items	OData Entities	Description
		A graphical information element is related to a dynamic document in the Dynamic Document Management domain. The dynamic document must contain graphical content.
Document information element	`DocumentInformationElement`	A document content object in an information structure or publication structure. A document information element is related to a document in the Document Management domain. The document can be of any format, for example, Microsoft Word, PDF, and so on.
Part List information element	`PartListInformationElement`	A content object of type parts list in an information structure or publication structure. A parts list information element is related to a parts list in the Parts List Management domain.
Generic information element	`GenericInformationElement`	A node in an information structure or publication structure, which points to other information elements. The applicability rules determine which information elements are included while publishing.
Table of Contents	`TableOfContent`	A content object of type table of contents.

Items	OData Entities	Description
		Table of contents object in a publication structure indicates that the published output should contain a table of contents at the specified location in the document.
Indexes	`Indexes`	A content object of type index. Index object in publication structure indicates that the published output should contain an index at the specified location in the document.
Transversal in document	`SIMDocument`	Supports transversal to the information element for a document.
Transversal in dynamic document	`SIMDynamicDocument`	Supports transversal to the information element for a dynamic document.

Action Available in the PTC Service Information Management Domain

The following action is available in the PTC Service Information Management domain:

GetStructure()

The action `GetStructure()` expands a publication and information structure, and retrieves its contents. The action is bound to the entity `NavigationCriteria`, that is, to the filter saved in Windchill. The action is available for `InformationStructure`, `InformationGroup`, `PublicationStructure`, `PublicationSection`, and `GenericInformationElement` entities.

When you call the `GetStructure()` action, in the request body of the URL, you must specify the ID of the `NavigationCriteria`. This is the ID of the saved filter you want to use as the filter criteria. If you do not specify the ID of the filter in the request body, then the default filter is used to work with the structure.

PTC Quality Domains

This section explains the Quality domains provided by PTC. These domains are available only if you have installed the relevant Quality products during Windchill installation.

PTC Quality Management System Domain

PTC Quality Management System (QMS) domain provides access to the people and places administration capability of Windchill. The domain enables you to create and manage people and places for a quality context.

The following table lists the significant OData entities available in the PTC Quality domain. To see all the OData entities available in the PTC Quality domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Note

The `GetDriverProperties()`, `GetPregeneratedValue()`, and the two `GetConstraints()` functions are not supported, though they are available in the metadata of the domain.

Items	OData Entities	Description
Quality	`Quality`	The `Quality` entity represents a Quality container.

Person or place	`PersonPlace`
	`Place`
	`BusinessUnit`
	`RegulatoryAgency`
	`Manufacturer`
	`Supplier`
	`Distributor`
	`FacilityHospital`
	`Person`
	`MedicalProfessional`
	`Patient`
		The `PersonPlace` and all the subentities represent a person or place which is related to a `Quality` entity.

Address	`Address`	The `Address` entity

Items	OData Entities	Description
		represents the addresses associated with a person or place.
Email	`Email`	The `Email` entity represents the email addresses associated with a person or place.
Phone number	`PhoneNumber`	The `PhoneNumber` entity represents the phone numbers associated with a person or place.
Cross reference	`Xreference`	The `Xreference` entity represents cross referenced items, such as, ERP system IDs or medical record numbers, that are associated with a person or place.
Relationships	`Relationship`	The `Relationship` entity represents the relationships between people or places.

Items	OData Entities	Description

Subject	`Subject`
	`Document`
	`PartInstance`
	`Part`
		The `Subject` entity represents the subject of a Quality entity. Subject is the object on which you want to perform an action. The `Subject` entity can be extended to include any changeable Windchill object such as, document, part and part instance.

Quality contact	`QualityContact`
	`PatientContact`
	`RegulatoryAgencyContact`
	`BusinessUnitOrOfficeContact`
	`MedicalProfessionalContact`
	`SupplierContact`
	`ManufacturerContact`
	`FacilityContact`
		These entities represent a person or place that can be linked to a Quality entity. The entities are not directly used in the PTC Quality domain. However, they can be used in other domains that extend the Quality domain.

PTC Nonconformance Domain

PTC NC (Nonconformance) domain provides access to the Windchill Nonconformance capabilities. The domain enables you to manage nonconformances. A nonconformance occurs when a product, manufacturing material, process, document, or other item does not conform to specifications.

The following table lists the significant OData entities available in the PTC Nonconformance domain. To see all the OData entities available in the PTC Nonconformance domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

The PTC NC domain extends the PTC Quality domain.

Items	OData Entities	Description
Nonconformance	`Nonconformance`	The `Nonconformance` entity represents a nonconformance process in Windchill.
Originating location	`Place`	The `Place` entity represents the location where the nonconformance originated. For example, the manufacturing site.
Originated by	`User`	The `User` entity represents the user who created the new nonconformance.
Related personal or location	`QualityContact`	The `QualityContact` entity represents a person or place relevant to the nonconformance request.
Affected objects	`AffectedObject`	The `AffectedObject` entity represents the subject that is affected by the nonconformance. This entity allows navigation to the Quality subject.
Other items	`OtherItem`	The `OtherItem` entity represents the objects that are stored outside of Windchill that are involved in the nonconformance. These items are not available as Quality subject.

PTC Customer Experience Management Domain

PTC Customer Experience Management (CEM) domain provides access to the Windchill customer experience capabilities. The domain enables you to manage customer experiences in their given workflow state. You can collect, document, track, trend, and report product quality issues recorded by customers as customer experiences.

The following table lists the significant OData entities available in the PTC Customer Experience domain. To see all the OData entities available in the PTC Customer Experience Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

The PTC Customer Experience Management domain extends the PTC Quality domain.

Items	OData Entities	Description
Customer experience	`CustomerExperience`	The `CustomerExperience` entity represents the customer experience process in Windchill.
Related products	`RelatedProduct`	The `RelatedProduct` entity represents a subject which is impacted by the customer experience. This entity allows navigation to the Quality subject.
Related personal or location	`QualityContact`	The `QualityContact` entity represents a person or place that is relevant to the customer experience request.

PTC Regulatory Master Domain

PTC Regulatory Master (RegMstr) domain provides access to the regulatory submission capabilities of Windchill. The domain enables you to create and manage regulatory submissions.

To be authorized to sell products in a specific geography, most companies are required to provide a qualification submission. The regulatory submission feature provides a centralized mechanism to track, manage, and maintain artifacts that are submitted to regulatory agencies.

Note

The PTC Regulatory Master (RegMstr) domain is not available in the Windchill 11.1-M020-CPS05 release.

The following table lists the significant OData entities available in the PTC Regulatory Master domain. To see all the OData entities available in the PTC Regulatory Master domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

The PTC Regulatory Master domain extends the PTC Quality domain.

Items	OData Entities	Description
Regulatory submission sent to a regulatory agency	`RegulatorySubmission`	The `RegulatorySubmission` entity represents a version of the regulatory submission. In Windchill, the `RegulatorySubmission` class is used to work with regulatory submissions. You can create subtypes for every type of regulatory submission that is sent to a regulatory agency.

Action Available in the PTC Regulatory Master Domain

The following action is available in the PTC Regulatory Master domain:

CreateFollowup

The `CreateFollowup` action creates a new iteration of the regulatory submission. You can create follow-ups for a completed or expired regulatory submission. After creating a follow-up task, the new iteration of the regulatory submission undergoes the same lifecycle to be submitted to the agency.

PTC CAPA Domain

PTC CAPA domain provides access to the Windchill CAPA capabilities. The domain enables you to manage and view corrective and preventive action objects (CAPAs) in their given workflow state.

The following table lists the significant OData entities available in the PTC CAPA domain. To see all the OData entities available in the PTC CAPA domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

The PTC CAPA domain extends the PTC Quality domain.

Items	OData Entities	Description
CAPA	`CAPA`	The `CAPA` entity represents the CAPA process in Windchill.
Sites	`CAPASite`	The `CAPASite` entity represents the locations which are impacted by CAPA. For example, the manufacturing site.
Affected Object	`AffectedObjects`	The `AffectedObject` entity represents the subject which is affected by CAPA. This entity allows navigation to the Quality subject.
Related personal or location	`QualityContact`	The `QualityContact` entity represents a person or place that is relevant to the CAPA request.
Plan	`Plan`	The entity `Plan` represents a plan state in CAPA, where further actions are decided.
Action	`Action`	The entity `Action` represents the actions of the CAPA plan.

PTC Audit Domain

PTC Audit domain provides access to the auditing capabilities available in the Quality product of Windchill. The owner of a quality container can create an audit. This domain enables you to retrieve the audits along with the audit details.

The following table lists the significant OData entities available in the PTC Audit domain. To see all the OData entities available in the PTC Audit domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

The PTC Audit domain extends the PTC Quality domain.

Items	OData Entities	Description
Audit	`Audit`	The `Audit` entity represents an audit. The entity supports only GET operation.
Audit details	`AuditDetail`	The `AuditDetail` entity represents an audit criterion, which is an individual audit question. The entity supports GET and PATCH operations.

PTC Info*Engine System Domain

PTC Info*Engine System domain provides access to the Info*Engine tasks of Windchill. Info*Engine enables you to access, manage, and present data from different information systems. The Info*Engine tasks help in retrieval and manipulation of data within the Info*Engine environment.

The domain provides an unbound function `InvokeIETask` to invoke the Info*Engine tasks on Windchill. Info*Engine tasks that take their input from web based forms or as URL parameters can be called with this function. Refer to the example Invoking an Info*Engine Task on page 184, for more information.

The following table lists the significant OData entities available in the PTC Info*Engine System domain. To see all the OData entities available in the PTC Info*Engine System domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Info*Engine output group	`Group`	The `Group` entity represents the XML output of an Info*Engine task. The entity consists of a collection of elements, where every element is a collection of attributes. An attribute is represented with a name-value pair.

PTC Factory Domain

The PTC Factory domain provides access to the manufacturing data management capabilities of Windchill. It also supports manufacturing process management (MPM) processes, which enable concurrent and collaborative development of product designs and manufacturing processes. The domain is available only if you install Windchill MPMLink.

The following table lists the significant OData entities available in the PTC Factory domain. To see all the OData entities available in the PTC Factory domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Standard operation	`StandardOperations`	The `StandardOperations` entity represents the version of a standard operation. In Windchill, use the `MPMStandardOperation` class to work with the versions of standard operations.

Control characteristic linked to a standard operation	`SOPSCCLinks`
	`DDLinks`
	`ResourceLinks`
	`SPLinks`
		`SOPSCCLinks` is an OData entity that represents the association between a standard operation and standard control characteristic. It contains attributes, such as, `ModelItemUID` and `GraphicData`.
		`DDLinks` is an OData entity that represents the association between a control characteristic, which is linked to a standard operation and describe documents.
		`ResourceLinks` is an OData entity that represents the association between a control characteristic, which is linked to standard operation and processing resources.
		`SPLinks` is an OData entity that represents the association between a control characteristic, which is linked to a standard operation and standard procedures.

Items	OData Entities	Description

Standard control characteristic	`StandardControlCharacteristics`
	`SCCDDLinks`
	`SCCSPLinks`
	`SCCResourceLinks`
		The `StandardControlCharacteristics` entity represents the version of a control characteristic. In Windchill, use the `MPMStandardCC` class to work with the versions of standard control characteristics.
		`SCCDDLinks` is an OData entity that represents the association between a standard control characteristic and describe documents.
		`SCCSPLinks` is an OData entity that represents the association between a standard control characteristic and standard procedure.
		`SCCResourceLinks` is an OData entity that represents the association between a standard control characteristic and processing resources.

PTC Manufacturing Process Management Domain

The PTC Manufacturing Process Management (MfgProcMgmt) domain provides access to the manufacturing process management capabilities (MPM) of Windchill. Manufacturing Process Management is the process of defining and managing the manufacturing processes, which are used to make parts, assemble final products, and perform inspections. The domain provides OData entities that represent business objects such as process plan, operation, and bill of process (BOP). The domain is available only if you install Windchill MPMLink.

The PTC Manufacturing Process Management domain references the PTC Document Management domain to provide navigation to allocated parts and operated on parts.

The following table lists the significant OData entities available in the PTC Manufacturing Process Management domain. To see all the OData entities available in the PTC Manufacturing Process Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Process Plan	`ProcessPlan` `StandardProcedure`	The `ProcessPlan` entity represents the version of a process plan. In Windchill, the `MPMProcessPlan` and `MPMProcessPlanMaster` classes are used to work with process plan versions. The `StandardProcedure` entity is derived from the `ProcessPlan` entity. `StandardProcedure` represents an instance of `MPMProcessPlan` class, where `standard` attribute is set to `true`.
Operation	`Operation`	The `Operation` entity represents the version of an operation. In Windchill, the `MPMOperation` and `MPMOperationMaster` classes are used to work with operation versions.
Sequence	`Sequence`	The `Sequence` entity represents the version of a sequence. In Windchill, the `MPMSequence` and `MPMSequenceMaster` classes are used to work with sequence versions.

Items	OData Entities	Description

Bill of Process	`BOP`
	`OperationHolder`
	`OperationHolderUsageLink`
		The `BOP` entity represents the process plan structure. `BOP` is an operation structure, which is expanded to the required number of levels.
		The `OperationHolderUsageLink` entity represents the association between parent and child parts in `OperationHolder` entities.
		In Windchill, the `MPMOperationUsageLink`, `MPMSequenceUsageLink`, and `MPMStandardProcessLink` classes are used to work with parent-child associations.

Resource	`WorkCenter`
	`Skill`
	`Tooling`
	`ProcessingMaterial`
		The `WorkCenter`, `Skill`, `Tooling`, and `ProcessingMaterial` entities represent the MPM resources. In Windchill, resources represent objects such as, personnel, material, equipment and so on, that perform the production activities. Manufacturing resources are the resources needed on the shop floor during the production, maintenance, inspection, or repair of parts.

Control Characteristic	`StandardControlCharacteristic`	The `StandardControl`

Items	OData Entities	Description
		`Characteric` entity represents a control characteristic version. In Windchill, the `MPMStandardCC` class is used to work with control characteristics.

subtypeable and softattributable Attributes

The PTC Manufacturing Process Management domain supports the `subtypeable` and `softattributable` attributes of Windchill. All the PTC Manufacturing Process Management domain entities that are backed by a persistable which implements the `Typed` interface, support these attributes.

Note

The `subtypeable` and `softattributable` attributes are not supported for the `ProcessPlan` entity. However, you can add the subentities of the `ProcessPlan` entity by explicitly configuring the entities in the domain.

Actions Available in the PTC Manufacturing Process Management Domain

The following actions are available in the PTC Manufacturing Process Management domain:

GetBOP

The action `GetBOP` returns the bill of process (BOP) for the process plan structure. The action is bound to the entity `NavigationCriteria`, that is, to the filter saved in Windchill.

When you call the `GetBOP` action you can specify the IDs of two `NavigationCriteria`, that is, `processPlanNavigationCriteriaId` and `relatedAssemblyNavigationCriteriaId` in the request body. These are the IDs of the saved filters you want to use as the filter criteria. If you do not specify the IDs of the filter in the request body, then the default filters are used to work with the process plan structure.

GetBOPWithInlineNavCriteria

The `GetBOPWithInlineNavCriteria` action returns the bill of process (BOP) for the process plan structure for the specified navigation criteria. Pass `processPlanNavigationCriteria` and `relatedAssemblyNavigationCriteria` as the input parameters.

In the navigation criteria, if the property `ApplyToTopLevelObject` is set to `True`, and no qualifying version is found for the top-level object, an error message is returned.

GetConsumed

The `GetConsumed` action returns the object associated to a consuming operation for the specified navigation criteria. Pass `processPlanNavigationCriteriaId` and `relatedAssemblyNavigationCriteriaId` as the input parameters.

GetConsumedWithInlineNavCriteria

The `GetConsumedWithInlineNavCriteria` action returns the object associated to a consuming operation for the specified navigation criteria. Pass `processPlanNavigationCriteria` and `relatedAssemblyNavigationCriteria` as the input parameters.

In the navigation criteria, if the property `ApplyToTopLevelObject` is set to `True`, and no qualifying version is found for the top-level object, an error message is returned.

PTC Change Management Domain

The PTC Change Management domain provides access to the change management capabilities of Windchill. The domain includes entities that represent business objects such as, problem report, variance, change request, change notice, and so on. The domain provides navigation to associated process objects and associated reference objects.

The following table lists the significant OData entities available in the PTC Change Management domain. To see all the OData entities available in the PTC Change Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Problem report	`ProblemReport`	The `ProblemReport` entity represents the version of a problem report. In Windchill, the `WTChangeIssue` and `WTChangeIssueMaster` classes are used to work with versions of a problem report.
Variance	`Variance`	The `Variance` entity represents the version of a variance. In Windchill, the `WTVariance` and `WTVarianceMaster` classes are used to work with versions of a variance.
Change request	`ChangeRequest`	The `ChangeRequest` entity represents the version of a change request. In Windchill, the `WTChangeRequest2` and `WTChangeRequest2Master` classes are used to work with versions of a change request.
Change notice	`ChangeNotice`	The `ChangeNotice` entity represents the version of a change notice. In Windchill, the `WTChangeOrder2` and `WTChangeOrder2Master` classes are used to work with versions of a problem report.
Associated process links	`ProcessLinks`	The `ProcessLinks` entity represents process links between a change object and its associated

Items	OData Entities	Description
		change objects. A process link contains only the `description` attribute and provides information about the attribute. **Note** This entity is supported only for flexible change links.
Associated reference links	`ReferenceLinks`	The `ReferenceLinks` entity represents reference links between a change object and its associated change objects. A reference link contains only the `description` attribute and provides information about the attribute.
Associated Process Objects	`ProcessObjects`	The `ProcessObjects` entity represents associated change objects with parent and child relationships for the given change object. **Note** This entity is supported only for flexible change links.
Associated reference objects	`ReferenceObjects`	The `ReferenceObjects` entity represents associated reference objects with parent and child relationships for the given change object. These are references to other change objects.

The following navigation properties are available:

- `AffectedLinks`—Links between change objects and their affected objects.
- `AffectedObjects`—Affected objects that are associated with the change objects.
- `AffectedByLinks`—Links between change objects and their affected objects.
- `AffectedByObjects`—Change objects that have affected the specified object.
- `Attachments`—Attachments associated with the change objects. You can see the contents of an attachment and download it.
- `ResultingObjects`—Resulting objects that are associated with change notices either through resulting links or unincorporated links.
- `ResultedByObjects`—Change task information for the specified resulting objects, resulting links and unincorporated links.
- `ResultingLinks`—Resulting links information for the specified change notice.
- `ResultingByLinks`—Resulting links information for the specified resulting object.
- `UnincorporatedLinks`—Unincorporated (hanging change) links information for the specified change notice.
- `UnincorporatedByLinks`—Unincorporated (hanging change) links information for the specified resulting object.
- `ChangeAdministratorI`—User details about change administrator I.
- `ChangeAdministratorII`—User details about change administrator II.
- `VarianceOwners`—User details about variance owner.

The following navigation properties support multiple objects:

- `ProcessLinks`—Process links for all the change objects.
- `ProcessObjects`—Process objects for all the change objects.
- `ReferenceLinks`—Reference links for all the change objects.
- `ReferenceObjects`—Reference objects for all the change objects.

PTC Classification Structure Domain

The PTC Classification Structure (ClfStructure) domain provides access to classification nodes and the hierarchy of classification nodes in Windchill. In this domain, you can perform a classification search. The domain includes entities that represent classification node and classified object.

The domain provides navigations to the child nodes or parent node of a classification node. It also provides navigation to the classified objects associated with a classification node.

The following table lists the significant OData entities available in the PTC Classification Structure domain. To see all the OData entities available in the PTC Classification Structure domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Classification node	`ClfNode`	The `ClfNode` entity represents a classification node. In Windchill, all the nodes which are accessible from the classification tree are represented using this entity. For example, the entity represents nodes accessed from classification administration, classification explorer, and so on. **Note** In the `ClfNodes.json` file, the `clfStructureNameSpace` property specifies the default namespace `com.ptc.csm.default_clf_namespace` for the classification node. If you want to use some other namespace, specify the namespace in the `clfStructureNameSpace` property. The property supports only one namespace.
Classified object	`ClassifiedObject`	The `ClassifiedObject` entity represents a classified object associated with a classification node.

In Windchill REST Services 1.3 or later, the property `Required` is added to the `ClfAttributeType` complex type. This property specifies if the attribute is a mandatory attribute for classification.

Support Classification Search Using ANY Operator with $filter Expression

You can use classification attributes to perform a classification search. To perform the search, use the `$filter` parameter along with the lambda operator ANY in the URL. You must query on the `ClassificationAttributes` property, which is available in the `ClassifiedObject` entity.

To query, follow these guidelines:

- In the query, you must specify the internal name, followed by the display value of the classification attribute. If both the parameters are not specified in the required sequence, the URL malformed exception is thrown.
- Enclose the name-value pair in parentheses.
- You can use the AND operator for the name-value pair.
- The `InternalName` property supports only EQ operator.
- Calls to methods `startswith`, `endswith`, and `contains` are supported for attributes, which are of type String. An exception is thrown for other attribute types.

The GET request URL for classification search is:

```
https://windchill.ptc.com/Windchill/servlet/odata/v1/ClfStructure/
ClfNodes('classificationNode')/ClassifiedObjects?$filter=
ClassificationAttributes/any(d:d/InternalName eq 'attributeInternalName'
and d/DisplayValue eq 'attributeDisplayValue')
```

For example, consider classification attribute with internal name `xje136` and display value `12`. To perform a classification search on the classification node `FASTENER-THREADEDINSERT`, use the following GET request:

```
GET
https://windchill.ptc.com/Windchill/servlet/odata/v1/ClfStructure/
ClfNodes('FASTENER-THREADEDINSERT')/ClassifiedObjects?$filter=
ClassificationAttributes/any(d:d/InternalName eq ‘xje136’
and d/DisplayValue eq '12') HTTP/1.1
```

Functions Available in the PTC Classification Structure Domain

The following functions are available in the PTC Classification Structure domain:

GetClassificationNodeInfo()

The function `GetClassificationNodeInfo()` returns the information about classification attributes for the specified classification node. The input parameters for the function are the namespace of the classification structure and internal name of the classification node.

The function returns the attribute details as a collection of `ClassificationInfo` complex type. Use this as a payload to classify an object.

GetClfBindingInfo()

The function `GetClfBindingInfo()` returns information about the classification binding attribute and the node associated with the classified object. The input parameters for the function are the OID of the classified object and namespace of the classification structure. If an object is not classified, then the function returns an empty response.

GetEnumTypeConstraintOnClfAttributes()

The function `GetSelectedTypesFromSavedSearch()` gets the legal or enumeration values for the specified classification attribute. In the response, the function returns a pair of internal name and display name for every legal or enumeration value.

PTC Saved Search Domain

The PTC Saved Search domain provides access to saved searches in Windchill. You can use the saved searches to execute a search.

The following table lists the significant OData entities available in the PTC Saved Search domain. To see all the OData entities available in the PTC Saved Search domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Saved Search	`SavedQuery`	The `SavedQuery` entity represents the saved searches in Windchill. You can use `$filter` expressions to query the attributes available in the entity. **Note** The attribute `ID` cannot be queried using the `$filter` expressions.

Saved searches are not access controlled. When you query for saved searches, you get a list of all the saved searches from Windchill, even if the saved search was not created by you, or shared with you.

You can share a saved search with members of a context, organization, or site. Such saved searches are set as global search. `GlobalSavedSearchVisibility` property represents global search. The information that appears in global searches for the `GlobalSavedSearchVisibility` property is access controlled, because context, organization, and site are access controlled. For global searches, if you do not have the required context, organization, and site level access, the search-related information appears as **Secured information**.

When you query or execute saved searches, the search queries are logged in the **Security Audit Reporting** utility in Windchill.

Functions Available in the PTC Saved Search Domain

The following functions are available in the PTC Saved Search domain:

GetSelectedTypesFromSavedSearch()

The function `GetSelectedTypesFromSavedSearch()` returns the object types, which were selected when the search was saved. This function is bound to the `SavedQuery` entity of an object.

ExecuteSavedSearch()

The function `ExecuteSavedSearch()` is used to execute a saved search. This function is bound to the `SavedQuery` entity of an object. The function supports pagination.

When you execute a saved search, you can override the keyword defined in the saved search. The keyword specified in the URL takes precedence over the keyword defined in the saved search. The override is applied only when the search is executed. The saved search is not updated with the keyword specified in the request URL.

When you use a saved search to execute a search, the result set may contain Windchill objects of various types. Some of the Windchill objects types are not available in Windchill REST Services. All the object types that are not available in Windchill REST Services are represented as `WindchillEntity`. For such Windchill objects, the function returns only the ID, created by, and last modified by attributes.

The object types, which are available in Windchill REST Services, are automatically mapped to the relevant entity type. The function returns all the information related to these objects in the search result.

PTC Visualization Domain

The PTC Visualization (Visualization) domain provides access to the visualization services of Windchill.

Use Windchill Visualization Services (WVS) to publish lightweight representations of native document content in Windchill. These lightweight representations are then stored in the Windchill database and can be viewed, managed and modified. You can open representations of Windchill documents, such as, CAD Documents, EPMDocuments, and WTDocuments, in Creo View. You can view and annotate the primary content and attachments of the documents in Creo View.

Use the PTC Visualization domain to download CAD data and view it in Creo View.

The following table lists the significant OData entities available in the PTC Visualization domain. To see all the OData entities available in the PTC Visualization domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Windchill representation	`Representation`	The `Representation` entity is a lightweight representation of CAD data that is stored in Windchill and is associated with parts and documents. The entity returns the following URLs, which you can use to start Creo View and download CAD data: • `CreoViewURL` • `3DThumbnailURL` • `2DThumbnailURL` • `AdditionalFiles` Refer to the section URLs Retrieved in the Representation Entity on page 133, for more information.

In addition to the entities, this domain also contains `RepresentationHyperlink` complex type, which retrieves additional information for a `Representation`. It contains a URL and additional attributes, such as, Object ID, Comments, Description, FormatIcon, CreatedOn, LastModified, FileName, FileSize, MimeType, Format and so on.

URLs Retrieved in the Representation Entity

Windchill Visualization Services converts CAD, XML, document, and other data formats into a neutral file format. The converted data is stored as a representation. You can view representations as thumbnail images, which are displayed on information pages of parts and listings throughout Windchill.

In Windchill REST Services, the `Representation` entity retrieves the following URLs. You can use these URLs to download the CAD data. The URLs are returned in the body of the response. Along with the URL, the entity also retrieves attributes such as, `FileSize`, `MimeType`, `FileName`, `Format`, and so on for 3D thumbnail, 2D thumbnail, and additional files.

- `CreoViewURL`—Contains the URL that starts the Creo View application.
- `3DThumbnailURL`—Contains the URL for 3D thumbnail.

- `2DThumbnailURL`—Contains the URL for 2D thumbnail.
- `AdditionalFiles`—Contains the URL to download additional files, which are non-native Creo View files. These files are associated with the specified representation.

For example, consider a `DerivedImage` with OID 786687. To retrieve the URLs for Creo View, thumbnail and additional files, send the following GET request:

```
GET /Windchill/servlet/odata/Visualization/Representations
('OR:wt.viewmarkup.DerivedImage:786687') HTTP/1.1
```

The function `GetDynamicStructureRepresentation()` returns a URL you can use to download the dynamic structure of a Creo View representation. Refer to the section on GetDynamicStructureRepresentation() on page 134, for more information.

Functions Available in the PTC Visualization Domain

The following functions are available in the PTC Visualization domain:

GetDynamicStructureRepresentation()

The function `GetDynamicStructureRepresentation()` returns a URL which you can use to download the dynamic structure of a Creo View representation. This function is bound to the `Representation` entity of an object. The function accepts navigation criteria *NavigationCriteriaId* as the input parameter. It is an optional parameter. The navigation criteria define which parts and objects are displayed in the dynamic structure. If the navigation criteria is not specified, the function uses the default navigation criteria to generate the URL for the dynamic structure.

You can use the product structure URL to load the CAD data in the WebGL viewer. The viewer uses Creo View WebGL Toolkit APIs.

GetFidelities()

You can get a list of fidelity values associated with a representation. The function `GetFidelities()` returns a list of fidelity values, which are represented as `PTC.EnumType`. The function retrieves the fidelity information in a value-display format.

GetPVZ()

You can get a PVZ file from a Creo View representation. The function `GetPVZ()` returns a ZIP file, which contains the OL, PVS, PVT, and other Creo View files. The PVZ file is created from the associated representation. The function is bound to the `Representation` entity of an object.

The function contains the following parameters:

- *IncludeAnnotations*—Includes or excludes annotations in the PVZ file. Pass the parameter as `true` to include annotations, when the PVZ file is created.
- *Fidelity*—This is an optional parameter. Specifies the value of fidelity.

Note

The function `GetPVZ()` does not retrieve multi-fidelity representations for releases prior to Windchill 11.1 M020.

PTC Product Platform Management Domain

The PTC Product Platform Management domain provides access to the Options and Variants capabilities of Windchill. The Options and Variants capabilities define fixed options and choices that are used to specify discrete configurations for a product. The domain provides OData entities that represent business objects such as options, choices, option sets, variant specifications, and so on.

Note

This domain is available only when you perform the following actions:

- You install Windchill PDMLink.
- In Windchill, the option `Configurable Module Support` is set to *Yes* in **Utilities ► Preference Management ► Options and Variants**.

The following table lists the significant OData entities available in the PTC Product Platform Management domain. To see all the OData entities available in the PTC Product Platform Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Options	`Option`	The `Option` entity represents a capability or feature of a product that can be designed with variations. An option may have several choices. The entity provides navigation to `Choice` or `OptionGroup` entities.
Choices in an option	`Choice`	The `Choice` entity represents the values that are associated with an option. This entity provides a navigation property that lists the options associated with the choice. You can also expand the navigation property to list the details of the option.
Collection of options	`OptionSet`	An `OptionSet` is a collection of options and choices. These options and choices are referenced from the option pool. This entity provides navigation property that lists all the options available in the option set. You can also expand the navigation property to list the choices available in each option.
Variant specification	`VariantSpecification`	Variant specification is a collection of inputs and selections specified for a configurable structure when you create a variant.

Information About Other Entities

The following entities are not a part of the service document.

- `OptionGroup`—Collection of options that enables you to organize the options available for a product.
- `OptionPoolItem`—Option groups and top-level options created for a Windchill product or library context. Every Windchill product or library context can have a pool of options that are used in option sets.
- `DesignOption`—Options created for product design.

  For example, the following URL retrieves design options:

  ```
  /ProdPlatformMgmt/Options/PTC.ProdPlatformMgmt.DesignOption
  ```
- `DesignChoice`—Choices created for product design.

  For example, the following URL retrieves design choices:

  ```
  /ProdPlatformMgmt/Choices/PTC.ProdPlatformMgmt.DesignChoice
  ```
- `SalesOption`—Options created for sales.

  For example, the following URL retrieves sales options:

  ```
  /ProdPlatformMgmt/Options/PTC.ProdPlatformMgmt.SalesOption
  ```
- `SalesChoice`—Choices created for sales.

  For example, the following URL retrieves sales choices:

  ```
  /ProdPlatformMgmt/Choices/PTC.ProdPlatformMgmt.SalesChoice
  ```
- `IndependentAssignedExpression`—Independent expressions that are assigned to objects. Use the navigation, `AssignedExpression`, to retrieve the details for independent expressions. Information about the expression aliases is also returned.

Action Available in the PTC Product Platform Management Domain

The following action is available in the PTC Product Platform Management domain:

GetAssignedExpressions

The `GetAssignedExpressions` action returns the assigned expressions for `Part`, `PartUse`, and `UsageOccurrence` entities. You can specify multiple objects as the input parameter. Basic and advanced types of expressions are supported.

The information related to assigned expressions is retrieved using the complex type, `AssignedExpression`.

To retrieve the assigned expressions for a single object, use the `GetAssignedExpression()` function defined in the PTC Product Management domain. See the section Function Available in the PTC Product Management Domain on page 85, for more information.

Both dependent and independent expression modes are supported.

PTC CAD Document Management Domain

PTC CAD Document Management domain provides access to the CAD data management capabilities of Windchill. CAD data management uses business objects, referred to as CAD documents to contain and manage CAD information in a Windchill database. A CAD document is a revision-controlled and lifecycle-managed object containing a CAD model or drawing file. The CAD model can be a file or a set of files containing information in a CAD application format. This domain enables you to get all the information related to a CAD document.

The following table lists the significant OData entities available in the PTC CAD Document Management domain. To see all the OData entities available in the PTC CAD Document Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description

CAD document	`CADDocument`
	`CADDocumentUse`
	`CADDocumentReference`
	`DerivedSource`
		The `CADDocument` entity represents a version of CAD document. Use the navigation property `AllPrimaryContents` to get details about the primary content

Items	OData Entities	Description
		associated with the CAD document. **Note** • `AllPrimaryContents` navigation property supports all the primary content that is directly downloaded in Windchill. It returns URLs to download the content. • The primary content that is not directly downloadable in Windchill and opens in other applications is not supported. For example, CAD assembly opens in Creo Parametric, and is not downloadable in Windchill. Such primary contents are not supported by the navigation property. • When the CAD document does not contain any primary content or contains primary content that is not supported by the navigation, the response returns an empty primary content. `"AllPrimaryContents": []`

Items	OData Entities	Description
		• The navigation `PrimaryContent` is not supported for `CADDocument` entity. In Windchill, the types with internal names, `DefaultEPMDocument` and `DefaultEPMDocumentMaster`, represent the versions of CAD document. Use the classes `EPMDocument` and `EPMDocumentMaster` to work with versions of CAD documents. The `CADDocumentUse` entity represents the link between a parent assembly and its components. It contains attributes such as, quantity, unit, location, and so on. In Windchill, `DefaultEPMMemberLink` type and `EPMMemberLink` class represent this association. The `CADDocumentReference` entity represents the link between a CAD document and its references. References are relationships between files that do not have a hierarchical or structural relationship. In Windchill, `DefaultEPMReferenceLink` type and

Items	OData Entities	Description
		`EPMReferenceLink` class represent this association. The `DerivedSource` entity represents the link between an image CAD document and its source. An image is an object that has copied most of its content from a source object. In Windchill, `EPMDerivedRepHistory` class represents this association.

Part associations	`PartAssociation`
	
	`BuildRuleAssociation`
	
	`BuildHistoryAssociation`
	
	`ContentAssociation`
		The `PartAssociation` entity represents the association links between a CAD document and a `WTPart`. The types of association links are further represented by the `BuildRuleAssociation`, `BuildHistoryAssociation`, and `ContentAssociation` entities.
		
		The `BuildRuleAssociation` and `BuildHistoryAssociation` entities represent the link between a CAD document and `WTPart` for all associations except Content. In Windchill, the `EPMBuildRule` and `EPMBuildHistory` classes represent this association.

Items	OData Entities	Description
		The `ContentAssociation` entity represents the link between a CAD document and `WTPart` for content association type. In Windchill, it is represented by `EPMDescribeLink` class.
CAD document structure	`CADStructure`	The `CADStructure` entity represents the CAD structure, which is expanded to required number of levels. Use the action `GetStructure` to retrieve the CAD structure.

Action Available in the PTC CAD Document Management Domain

The following action is available in the PTC CAD Document Management domain:

GetStructure

The action `GetStructure` returns a CAD structure. The action is bound to the `CADDocument` entity.

You can expand the `CADDocument` and `CADDocumentUse` navigation properties for more details of the documents.

When you call the `GetStructure` action in the request body of the URL, you can specify the ID of the `NavigationCriteria`. This is the ID of the saved filter you want to use as the filter criteria. If you do not specify the ID of the filter in the request body, then the default filter is used to work with the CAD structure. Alternatively, you can specify the navigation criteria in the request payload.

The action additionally returns an attribute `Resolved` in the response body. The attribute indicates if the link to the document is resolved in the configuration specification. The attribute returns the boolean value true or false depending on whether the link is resolved.

When you call the `GetStructure` action, the following additional URLs are returned:

- `PVTreeId` is the occurrence path of the CAD assembly to its member subcomponents in the viewable file. The complete path from the root of the BOM structure is returned. This URL can be used to work with Visualization tree. For example, in an application you consume this URL and highlight the component in the Visualization tree.
- `PVParentTreeId` is the occurrence path to the parent of the component part in the viewable file. The complete path from the root of the BOM structure is returned.

The action supports the parameter `BOMMembersOnly`. If you specify `BOMMembersOnly` as true in the request body, only the CAD documents that participate in the BOM structure are returned. The default value of `BOMMembersOnly` is false. If the request body is empty, the default value of the parameter is used, and all the structure members are returned.

Note

If the BOM structure contains documents that do not contribute to the BOM, and the user does not have access to such documents, then even though the `BOMMembersOnly` parameter is set to `true` in the request body and navigation criteria, the unresolved dependents are returned.

In the request body, use the following code to specify the `BOMMembersOnly` parameter:

```
{
"BOMMembersOnly" : true
}
```

The preferences set for the **Auto Associate** action in Windchill are honored by `GetStructure` while returning the structure. The document types and subtypes added in this preference do not participate in the BOM structure of the CAD document.

For example, CAD documents are not included in the structure in the following cases when `BOMMembersOnly` is set to true:

- If you add the document types and subtypes in the following preference:

  Utilities ► Preference Management ► Operation ► Auto Associate ► Disallow Structure CAD Document Types
- If the CAD document has a reusable attribute set in the following preference:

  Utilities ► Preference Management ► Operation ► Auto Associate ► Part Structure Override Attribute Name

PTC Event Management Domain

The PTC Event Management domain provides access to the webhook subscription capabilities of Windchill. Webhook subscriptions enable Windchill to send event notifications to external systems when certain events or actions occur on object, folder, or context in Windchill through a webhook URL. The domain must be used along webhook to subscribe to events.

Note

- Webhook subscription is not supported for Windchill soft types.
- Subscription to events using the PTC Event Management domain is supported only for entities available in the following domains:
  - PTC Product Management domain
  - PTC Document Management domain
  - PTC Data Administration domain
  - PTC Change Management domain

The following table lists the significant OData entities available in the PTC Event Management domain. To see all the OData entities available in the PTC Event Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Subscription to an event	`EventSubscription`	The `EventSubscription` entity represents the subscription to an event. Specify the URL to the external system in the `CallbackURL` property. Notification of the event is sent to this URL. To create a new subscription, use the POST operation. See the examples in the section Examples for the PTC Event Management Domain on page 198 for more information on how to specify the payload while creating a subscription. To delete a subscription, use the DELETE operation. The `EventSubscription` entity has a navigation property `SubscribedEvent` that enables you to see the subscribed event.
Events	`Event`	The `Event` entity represents the Windchill events to which you can subscribe.
Subscription to events that occur on a Windchill object	`EntityEventSubscription`	The `EntityEventSubscription` entity represents the subscription to events that occur on a Windchill object.

Items	OData Entities	Description
		The entity has a navigation property `SubscribedOnEntity` that enables you to see the entity instance for the subscription.
Subscription to events that occur on a container	`EntityTypeInContainerEventSubscription`	The `EntityTypeInContainerEventSubscription` entity represents the subscription to events that occur on a type of Windchill object in the specified context. For example, a Windchill product or library can serve as a context for a subscription. The entity has a navigation property `SubscribedOnContext` that enables you to see the context of the subscription.
Subscription to events that occur on a folder	`EntityTypeInFolderEventSubscription`	The `EntityTypeInFolderEventSubscription` entity represents the subscription to events that occur on a type of Windchill object in the specified folder. The entity has a navigation property `SubscribedOnFolder` that enables you to see the folder that serves as a context of the subscription.

Function Available in the PTC Event Management Domain

The following function is available in the PTC Event Management domain:

GetApplicableEvents

The function `GetApplicableEvents` returns a list of all the events that are available for subscription for the specified Windchill object. Specify the Windchill object in the format `PTC.<Domain_Name>.<Entity_Type>`. For example, to specify a Part, pass the information as `PTC.ProdMgmt.Part` in the URL:

```
GET /Windchill/servlet/odata/EventMgmt/GetApplicableEvents
(EntityName='PTC.ProdMgmt.Part')
```

PTC Supplier Management Domain

The PTC Supplier Management domain provides access to the supplier management capabilities of Windchill. Supplier management enables companies to integrate and manage supply chain data in Windchill. You can create supplier organizations in Windchill PDMLink and populate the database with manufacturer and vendor parts. You can also create and update Approved Manufacturer Part Lists (AMLs) and Approved Vendor Part Lists (AVLs). The AMLs and AVLs are created using the sourcing contexts. PTC Supplier Management domain is available only if you install Supplier Management module in Windchill.

The PTC Supplier Management domain enables you to read sourcing contexts. Other supplier management objects such as, supplier part, manufacturer part, vendor part, and AXLEntry are available in the PTC Product Management domain. See the section PTC Product Management Domain on page 76, for more information.

The following table lists the significant OData entities available in the PTC Supplier Management domain. To see all the OData entities available in the PTC Supplier Management domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Sourcing context (sourcing relationship)	`SourcingContext`	The `SourcingContext` entity represents the relationship between sourcing contexts in Windchill. Souring context is created in an organization and is used to create an AML or AVL in a specific context.

The `SourcingContext` entity and `SourcingStatus` complex type support filtering using `$filter` parameter along with the lambda operator ANY in the URL.

For more information on how to use lambda operator, see the guidelines explained in the section Support Classification Search Using ANY Operator with $filter Expression on page 129.

PTC Workflow Domain

PTC Workflow (Workflow) domain provides access to the workflow capabilities of Windchill. A workflow enables you to automate procedures in which information, tasks, and documents are passed to several participants, maybe across multiple companies.

The domain enables you to get all the information related to a workflow task that is associated with a user. Use this domain to perform the following actions:

- Get the list of states of the workitems.
- Get the workitems assigned to you.
- Save tasks.
- Complete tasks.
- Get the list of users to whom a specified task can be reassigned.
- Reassign your tasks to other users.

The following table lists the significant OData entities available in the PTC Workflow domain. To see all the OData entities available in the PTC Workflow domain, refer to the EDM of the domain. The domain EDM is available at the metadata URL.

Items	OData Entities	Description
Work item in Windchill	`WorkItem`	The `WorkItem` entity represents tasks in Windchill. A work item is a task that is assigned to a user.
Subject of a workflow	`Subject`	The `Subject` entity represents a business object that is associated with the `WorkItem` entity. `Subject` can be any business object that is lifecyle managed. For example, parts, CAD documents, change requests, and so on.

Items	OData Entities	Description
Activity in workflow	`Activity`	`Activity` represents an action in a process which is assigned in the form of tasks or workitems to users. Every `WorkItem` entity is associated with `Activity`.
Templates in a workflow	`WorkItemProcesssTemplate`	The `WorkItemProcesssTemplate` entity represents a workflow template. It is represented as `WfProcessTemplate` object in Windchill. The `Workitem` entity is associated with a workflow template.

Items	OData Entities	Description

Information about the user	`Owner`
	`CompletedBy`
	`OriginalOwner`
		The `Owner`, `CompletedBy`, and `OriginalOwner` entities represent the `WTUser` object in Windchill. Owner information is available for every `WorkItem`. The `CompletedBy` and `OriginalOwner` entities are populated using the complete, reassign, or delegate action.

Auditing details of a task	`VotingEventAudit` `WfEventAudit`	These entities represent `WorkItem` that is associated with the `Activity` entity. The `WfEventAudit` and `VotingEventAudit` entities contain the audit information for a specific task. They are associated with work items that are complete.

Actions Available in the PTC Workflow Domain

The following actions are available in the PTC Workflow domain:

CompleteWorkitem

The `CompleteWorkitem` action completes the specified task. The action is bound to the `Workitem` entity.

You must provide the following mandatory parameters as input to the action in the request body:

```
{
  "UserEventList":[<routing_options >],
"WorkitemComment":"<comment_for_the_workitem>",
"VoteAction":"<Vote_option_selected_by_the_participant>",
"AutomateFastTrack":<option_to_specify_if_change_notice_should_be_
explicitly_created, valid values true or false>,
```

```
"Variables":[<workflow_activity_variables>]
}
```

SaveWorkitem

The `SaveWorkitem` action saves the specified task. The action is bound to the `Workitem` entity.

You must provide the following mandatory parameters as input to the action in the request body:

```
{
  "UserEventList":[<routing_options >],
"WorkitemComment":"<comment_for_the_workitem>",
"VoteAction":"<Vote_option_selected_by_the_participant>",
"AutomateFastTrack":<option_to_specify_if_change_notice_should_be_
explicitly_created, valid values true or false>,
"Variables":[<workflow_activity_variables>]
}
```

ReassignWorkItems

The `ReassignWorkItems` action reassigns the work items or tasks to a specified user. This is an unbound action.

You must provide the following mandatory parameters as input to the action in the request body. You can pass the corresponding entities or entity IDs as input parameters.

- *WorkItems*—List of work item entities.

> **Note**
>
> While reassigning the work items, if you pass all the attributes of a work item as input, the action only considers the work item ID attribute. The other work item attributes are ignored.

- *User*—User entity.

```
{
     "WorkItems": [{"ID":"<workitem_entity_ID>"}],
     "User": {"ID":"<user_entity_ID>"},
  }
```

Function Available in the PTC Workflow Domain

The following function is available in the PTC Workflow domain:

GetWorkItemReassignUserList

The `GetWorkItemReassignUserList` function returns a list of valid users to whom the specified work items can be reassigned.

Accessing Domains

This section describes how clients work with domains. The information in this section applies to all the domains installed with Windchill REST Services.

When Windchill REST Services is installed, the servlet `WcRestServlet` is enabled in the Windchill installation. All requests to the domain resources, which are enabled by Windchill REST Services, pass through this servlet. The root URL to access this servlet is `https://<Windchill server>/Windchill/servlet/odata/`. When an HTTP GET request is sent to this URL, the servlet responds back with a list of domains available on the server.

From the servlet response, clients can select a domain, and send a GET request to the root URL of the domain to get a list of available entity sets. Clients can send GET, POST, PUT, and DELETE requests to the URLs of the entity sets.

The following URLs are used to interact with Windchill REST Services:

- REST Root URL—`https://<Windchill server>/Windchill/servlet/odata/`

  A GET request to this URL lists the domains available on the Windchill server. Clients can use this URL to get the list of services that are available on a Windchill server.

  For example, the output for the following request is as shown below:

  Request URL:

  ```
  GET https://windchill.ptc.com/Windchill/servlet/odata
  ```

  Output:

  ```
  [
   {
   "path": "https://windchill.ptc.com/Windchill/servlet/odata/v1/PrincipalMgmt",
   "name": "PTC Principal Management Domain",
   "description": "PTC Principal Management Domain",
   "id": "PrincipalMgmt"
   },
   {
   "path": "https://windchill.ptc.com/Windchill/servlet/odata/v1/DataAdmin",
   "name": "PTC Data Administration Domain",
   "description": "PTC Data Administration domain",
   "id": "DataAdmin"
  ```

```
},
{
"path": "https://windchill.ptc.com/Windchill/servlet/odata/v1/ProdMgmt",
"name": "PTC Product Management Domain",
"description": "PTC Product Management Domain",
"id": "ProdMgmt"
},
{
"path": "https://windchill.ptc.com/Windchill/servlet/odata/v1/DocMgmt",
"name": "PTC Document Management Domain",
"description": "PTC Document Management Domain",
"id": "DocMgmt"
},
{
"path": "https://windchill.ptc.com/Windchill/servlet/odata/v1/PTC",
"name": "PTC Common Domain",
"description": "PTC Common Domain",
"id": "PTC"
}
]
```

- Domain Root URL—`https://<Windchill server>/Windchill/servlet/odata/<Domain>`

  A GET request to this URL returns the information about the entity sets available in the domain. This request is same as that defined in OData protocol for the Service Root URL. The `<Domain>` in the URL refers to the domain identifier `id`, which is returned by the REST Root URL in the list of domains.

  For example, the output for a GET request to the Domain Root URL of the `ProdMgmt` domain is as shown below. The output shows that the `ProdMgmt` domain contains entity sets such as, `Parts`, `Containers` and so on.

  Request URL:

  ```
  GET https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/
  ```

  Output:

  ```
  {
      "@odata.context": "$metadata",
      "value": [
          {
              "name": "Parts",
              "url": "Parts"
          },
          {
  ```

```
            "name": "Containers",
            "url": "Containers"
        },
        {
            "name": "Representations",
            "url": "Representations"
        },
        {
            "name": "Documents",
            "url": "Documents"
        }
    ]
}
```


- Domain Metadata URL—`https://<Windchill server>/Windchill/servlet/odata/<Domain>/$metadata`

  A GET request to this URL returns the entity data model for the domain defined in the Common Schema Definition Language (CSDL). In the OData protocol, this is called the Metadata Document URL.

  For example, the output from a GET request to this URL for the `PrincipalMgmt` domain is as shown below. This URL is used to get information about the entity sets, entities, and entity relations provided by the service. For more information on entity sets, entities, and entity relations, please refer to the OData protocol documentation.

  Request URL:

  ```
  GET https://windchill.ptc.com/Windchill/servlet/odata/PrincipalMgmt/$metadata
  ```

  Output:


```
<?xml version='1.0' encoding='UTF-8'?>
<edmx:Edmx Version="4.0" xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
 <edmx:Reference Uri="https://windchill.ptc.com/Windchill/servlet/odata/v1/PTC/$metadata">
        <edmx:Include Namespace="PTC"/>
    </edmx:Reference>
    <edmx:DataServices>
        <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="PTC.PrincipalMgmt">
            <EntityType Name="Group" BaseType="PTC.PrincipalMgmt.Principal">
                <Property Name="Description" Type="Edm.String">
                    <Annotation Term="PTC.ReadOnly"/>
                </Property>
                <Property Name="DomainName" Type="Edm.String">
                    <Annotation Term="PTC.ReadOnly"/>
                </Property>
```

```
                <Annotation Term="Core.Description">
                    <String>Groups</String>
                </Annotation>
                <Annotation Term="PTC.Operations">
                    <String>READ</String>
                </Annotation>
            </EntityType>
            <EntityType Name="User" BaseType="PTC.PrincipalMgmt.Principal">
                <Property Name="LastName" Type="Edm.String"/>
                <Property Name="FullName" Type="Edm.String"/>
                <Property Name="EMail" Type="Edm.String"/>
                <Property Name="UserDomain" Type="Edm.String"/>
                <Annotation Term="Core.Description">
                    <String>Users</String>
                </Annotation>
                <Annotation Term="PTC.Operations">
                    <String>READ</String>
                </Annotation>
            </EntityType>
<ComplexType Name="OrgId">
                <Property Name="CodingSystem" Type="Edm.String"/>
                <Property Name="UniqueIdentifier" Type="Edm.String"/>
                <Annotation Term="Core.Description">
                    <String>Organization identifier</String>
                </Annotation>
            </ComplexType>
            <EntityContainer Name="Windchill">
                <EntitySet Name="Representations" EntityType="PTC.Representation"/>
                <EntitySet Name="Groups" EntityType="PTC.PrincipalMgmt.Group"/>
                <EntitySet Name="Users" EntityType="PTC.PrincipalMgmt.User"/>
                <EntitySet Name="Principals" EntityType="PTC.PrincipalMgmt.Principal"/>
            </EntityContainer>
        </Schema>
    </edmx:DataServices>
</edmx:Edmx>
```

Entity Set URL—https://<Windchill server>/Windchill/servlet/odata/*<Domain>*/*<EntitySetURL>*

An Entity Set URL references an entity set, which is available in the response of a domain, to a GET request by the Domain Root URL.

In the Domain Root URL example above, you can see that there is an entity set named `Parts` that also has a `url` for `Parts`. The Entity Set URL is `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts`. A GET request to this URL returns a set of entities as shown below:

Request URL:

```
GET https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts
```

Output:


```
{
    "@odata.context": "https://windchill.ptc.com/Windchill/servlet/odata/v2/ProdMgmt/
       $metadata#Parts",
    "value": [
        {
            "ID": "OR:wt.part.WTPart:62850",
            "Name": "LOWER_SUPPORT",
            "Number": "GC000019",
            "EndItem": false,
            "TypeIcon": {
                "Path": "https://windchill.ptc.com/Windchill/wtcore/images/part.gif",
                "Tooltip": "Part"
            },
            "Identity": "GC000019, LOWER_SUPPORT, A (Design)",
            "GeneralStatus": null,
            "ShareStatus": null,
            "ChangeStatus": null,
            "Superseded": null,
            "AssemblyMode": {
                "Value": "separable",
                "Display": "Separable"
            },
            "DefaultUnit": "ea",
            "DefaultTraceCode": "0",
            "Source": "make",
            "ConfigurableModule": "standard",
            "GatheringPart": false,
            "PhantomManufacturingPart": false,
            "OwningDesignCenter": null,
            "OwningBusinessUnit": null,
            "view": "Design",
            "CheckoutState": "Checked in",
            "Comments": null,
```

```
            "State": {
                "Value": "INWORK",
                "Display": "In Work"
            },
            "LifeCycleTemplateName": "Basic",
            "VersionID": "VR:wt.part.WTPart:62849",
            "Revision": "A",
            "Version": "A.1 (Design)",
            "Latest": true,
            "CreatedOn": "2017-04-08T03:47:23Z",
            "LastModified": "2017-04-08T03:47:23Z"
        }
]
}
```

Entity Set URL is the main endpoint to perform create, read, update, and delete operations using the HTTP requests POST, GET, PATCH and DELETE respectively.

Let us continue using the above example. To create a part the client sends a POST request to the URL `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts`. The body of the request contains a set of property names and values specified in a format that is acceptable to the server. Some of the acceptable formats are JSON, XML, and so on.

To update the part, clients send a PATCH request on the same URL. The body of the PATCH request contains a set of property names and values that will be modified.

To delete a part, clients send a DELETE request to the URL `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts('<key>')`. In this URL, `key` is the unique identifier for the part in the entity set. The object reference string in Windchill is treated as the `key`. To delete a part with object reference '`OR:wt.part.WTPart:668899`', the DELETE request is `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:668899')`.

Clients usually interact with the REST APIs using the Entity Set URL. All the entities in a domain may not have entity sets. Therefore, some entities in the domain are available using navigations. For example, a specific `PartUse` entity in the ProdMgmt domain is accessed by `https://windchill.ptc.com/Windchill/servlet/odata/ProdMgmt/Parts(<part_key>)/Uses(<uses_key>)`. Here `<part_key>` and `<uses_key>` are object reference strings that uniquely identify a part and a usage link.

Actions Available for Single and Multiple Objects

Actions are available to support the following operations for single and multiple objects.

These following actions are available for PTC Product Management and PTC Document Management domains, for parts and documents respectively.

- Creating objects
- Updating objects
- Deleting objects
- Checking in objects—Available for all entities that inherit `workable` capability
- Checking out objects—Available for all entities that inherit `workable` capability
- Undoing a check out—Available for all entities that inherit `workable` capability

Action to create a revision of objects is available for all the entities of a domain that inherit the `versioned` capability.

Actions for Single Object

The following actions take a single object as input parameter.

- These actions are available in PTC Product Management and PTC Document Management domains.
  - `CheckIn`—Pass the following input parameters:
    - Part or document to check in depending on the domain
    - Check in note
    - KeepCheckedOut
    - CheckOutNote
  - `CheckOut`—Pass the following input parameters:
    - Part or document to check out depending on the domain
    - Check out note
  - `UndoCheckOut`—Pass as input parameter the part or document for which you want to undo the check out.
- `Revise` action is available for domains that inherit the `versioned` capability. Pass as input parameter the part or document ID that you want to revise. You can also pass other optional parameters such as `VersionId` if the preference **Allow Override On Revise** is set to `Yes`.

Actions for Multiple Objects

The following actions take a collection of objects as the input parameter.

- These actions are available for all the entities that inherit the `workable` capability.
  - `CheckIn<EntitySet_name>`
  - `CheckOut<EntitySet_name>`
  - `UndoCheckOut<EntitySet_name>`
- `Delete<EntitySet_name>`—Available for entities in a domain where `"multiOperations": "DELETE"` is specified in the `Entity JSON` file.
- `Revise<EntitySet_name>`—Available for entities in a domain that inherits the `versioned` capability.
- These actions are available for all the entities in the PTC Product Management and PTC Document Management domains.
  - `Create<EntitySet_name>`—Available for entities where `"multiOperations": "CREATE"` is specified in the `Entity JSON` file.
  - `Update<EntitySet_name>`—Available for entities where `"multiOperations": "UPDATE"` is specified in the `Entity JSON` file.

For example, the actions available in the PTC Product Management domain are as follows:

- `CreateParts`—Pass the following input parameters:
  - Collection of parts to create
  - ID of the container in which you want to create the parts and other required attributes to create a part
- `UpdateParts`—Pass the following input parameters:
  - Collection of parts to update
  - Attributes of part with updated values
- `CheckInParts`—Pass the following input parameters:
  - Collection of parts to check in
  - Check in note
  - KeepCheckedOut
  - CheckOutNote
- `CheckOutParts`—Pass the following input parameters:
  - Collection of parts to check out
  - Check out note

- `UndoCheckOutParts`—Pass as input parameter a collection of parts for which you want to undo the check out operation.
- `DeleteParts`—Pass as input parameter a collection of parts which you want to delete. When the objects to delete have iterations and revisions, the delete operation is executed as described in the following table:

Details of URL Payload	Delete Operation
Part entity specified in the payload is not the latest iteration in its revision	Part object is not deleted and delete operation is rolled back
Part entity specified in the payload is not the latest iteration in its revision, and later revisions of the part object exist but are not specified in the payload	Part object is not deleted and delete operation is rolled back
Part entity specified in the payload is the latest iteration in its revision, and all the latest revisions of the part entity are also specified in the payload	Deletes all iterations of all revisions
Part entity specified in the payload is the latest iteration of the latest revision	Deletes all iterations of the latest revision

- `ReviseParts`—Pass as input parameter a collection of parts that you want to revise.

Similarly, `CheckInDocuments`, `CheckOutDocuments`, and so on are available in PTC Document Management domain.

Execution of Actions on Multiple Objects

When you perform multiple objects actions, consider the following:

- You can perform the multiple object action on one or more objects.
- The action is successfully executed for all objects. If the action fails for any object in the collection, the entire action is rolled back, the operation stops, and an error message is returned to the caller.

Common Navigation Properties Available in All the Domains

Windchill REST Services provides the following navigations properties which contain all the information related to the user. The navigation properties are available in all the domains that import the PTC Principal Management domain and for all the Windchill entities that support the creator and modifier attributes.

- `Creator`—Information about user who created the entity.
- `Modifier`—Information about user who modified the entity.

The navigation properties enable filtering on the source entity for the following user attributes:

- User name
- Full name
- Last name
- Email address

Examples for Performing Basic REST Operations

Fetching a NONCE Token from a Service

This example shows you how to fetch a NONCE token from a service. Use the following GET request.

URI

```
https://<Windchill server>/Windchill/servlet/odata/PTC/GetCSRFToken()
```

The token is returned in the JSON response. For example, the response is as shown below:

```
{
 "@odata.context": "https://windchill.ptc.com/Windchill/servlet/odata/v1/PTC/
$metadata#CSRFToken",
 "NonceKey": "CSRF_NONCE",
 "NonceValue": "8q87WtSxvWkSH9FMtsQUboOI5TtCS7gWh8RUb4OG ="
}
```

The value of `CSRF_NONCE` returned from this request must be passed as request header in all the examples provided in this User's Guide to create (POST requests), modify (PUT and PATCH requests), or delete (DELETE request) entities.

Creating a Part

This example shows you how to create a part. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Name":"TestWTPart_001",

 "AssemblyMode": {
  "Value": "separable",
  "Display": "Separable"
 },

"PhantomManufacturingPart" : false,

"Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:48507000')"
}
```

Creating Multiple Parts

This example shows you how to create multiple parts. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/CreateParts HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Parts": [
    {
      "Name": "test1",
      "AssemblyMode":
      {"Value": "inseparable"}
     ,
      "DefaultUnit" :
      "Value": "kg"}
     ,
      "DefaultTraceCode":
      {"Value": "X"}
     ,
      "Source":
      {"Value":"buy"}
     ,
      "Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:302725')"
```

```
    },
    {
      "Name": "test2",
      "AssemblyMode":
      {"Value": "separable"}
    ,
      "DefaultUnit" :
      {"Value": "ea"}
    ,
      "DefaultTraceCode":
      {"Value": "L"}
    ,
      "Source":
      {"Value":"make"}
    ,
      "Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:302725')"
   }
  ]
}
```

Creating a Part in a Different Organization

This example shows you how to create a part in a different organization. In Windchill, set the preference **Expose Organization** in **Preference Management** utility to `Yes`. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Name":"TestWTPart_001",

 "AssemblyMode": {
  "Value": "separable",
  "Display": "Separable"
 },

"PhantomManufacturingPart" : false,

   "Organization@odata.bind": "Groups('OR:wt.org.WTOrganization:274098')",
"Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:48507000')"
}
```

Create a Part Usage Link with Occurrences

This example shows you how to create a part usage link with occurrences. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('VR:wt.part.WTPart:48796525')/Uses HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
 "Quantity" : 2,
 "Unit" : {
"Value": "ea",
    "Display": "Each"
  },
 "FindNumber" : "100",
 "LineNumber" : 100,
 "TraceCode": {
"Value": "0",
    "Display": "Untraced"
  },
 "Uses@odata.bind" : "Parts('OR:wt.part.WTPart:48796415')",
   "Occurrences": [
        {
            "ReferenceDesignator": "R1",
            "Location": {
                "PointX": 0,
                "PointY": 1,
                "PointZ": 1,
                "PointUnit": "m",
                "AngleX": 1.04,
                "AngleY": 1.04,
                "AngleZ": 1.04,
                "AngleUnit": "r"
            }
        },
        {
            "ReferenceDesignator": "R2",
            "Location":  {
                "PointX": 1,
                "PointY": 1,
                "PointZ": 0,
                "PointUnit": "m",
                "AngleX": 3.14,
                "AngleY": 3.14,
                "AngleZ": 3.14,
                "AngleUnit": "r"
            }
        }
    ]

}
```

Deleting a Part Usage Link

This example shows you how to delete a part usage link. Use the following DELETE request.

URI

```
DELETE /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:48796526')/
                Uses('OR:wt.part.WTPartUsageLink:48796528') HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Reading the Bill of Material (BOM)

This example shows you how to read the bill of material (BOM) for a product structure. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:44148884')/
   PTC.ProdMgmt.GetBOM?$expand=Components($expand=Part($select=Name,Number),
   PartUse,Occurrences;$levels=max) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{NavigationCriteria={"ID":"OR:wt.filter.NavigationCriteria:186213"}}
```

Reading the Bill of Material (BOM) Along with Path Details

This example shows you how to read the bill of material (BOM) for a product structure.

Use the following POST URI with the request body to retrieve the BOM with expand on occurrences.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:254405')
/PTC.ProdMgmt.GetPartStructure?$expand=Components($expand=Part,PartUse,Occurrence) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{NavigationCriteria={"ID":"OR:wt.filter.NavigationCriteria:186213"}}
```

Use the following POST URI with the request body to retrieve the BOM without occurrences.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:254405')/
PTC.ProdMgmt.GetPartStructure?$expand=Components($expand=Part,PartUse)  HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{NavigationCriteria={"ID":"OR:wt.filter.NavigationCriteria:48786407"}}
```

Querying the Part Using a Filter

This example shows you how to query a part using a filter. Use the following GET request.

URI for Filter Based on Soft Attribute

```
GET /Windchill/servlet/odata/ProdMgmt/Parts?$filter=contains(CustomAttribute,'value') HTTP/1.1
```

URI for Filter Based on Part Name

```
GET /Windchill/servlet/odata/ProdMgmt/Parts?$filter=Name eq 'TestWTPart_001' HTTP/1.1
```

Reading a Part by ID with Expanded Navigation

This example shows you how to read a part with its ID with expanded navigation. Use the following GET request.

URI for Part Uses Link with Expand Filter

```
GET /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:48796184')?$expand=Uses HTTP/1.1
```

URI for Part Uses Link and Its Occurrences with Expand Filter

```
GET /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:48796184')
?$expand=Uses($expand=Occurrences) HTTP/1.1
```

Checking Out a Part

This example shows you how to check out a part. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:48796184')/PTC.CheckOut HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"CheckOutNote" : "This is checkout note."
}
```

Checking In a Part

This example shows you how to check in a part. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:48796184')/PTC.CheckIn HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"CheckInNote" : "This is checkin note."
}
```

Revising Multiple Parts

This example shows you how to change the revision of multiple parts. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/ReviseParts HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "Parts":[
     { "ID": "OR:wt.part.WTPart:270209" }
     ,
     { "ID": "OR:wt.part.WTPart:270219" }
   ]
}
```

Retrieving the Components List for a Part Structure

This example shows you how to retrieve a components list for a part structure with the action `GetPartsList` and by expanding navigation properties `Part` and `PartUses`. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:12345')/PTC.ProdMgmt.GetPartsList?$
expand=Part,PartUses HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
Accept: application/json;odata.metadata=full
```

Updating the Common Attributes of a Part

This example shows you how to update the common attributes of a part. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/Parts('OR:wt.part.WTPart:918283')/
PTC.UpdateCommonProperties HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Updates": {
"Name":"NewName",
"Number":"NewNumber",
"DefaultTraceCode":
        {
"Value": "L",
  "Display": "Buy"
        }
}
}
```

Updating Multiple Parts

This example shows you how to update multiple parts in a single call. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/ProdMgmt/UpdateParts HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "Parts": [
     {
        "ID":"OR:wt.part.WTPart:2200087",
        "AssemblyMode":
        {"Value": "inseparable"}
        ,
        "DefaultUnit" :
        {"Value": "kg"}
        ,
        "DefaultTraceCode":
        {"Value": "L"}
     },
    {
       "ID":"OR:wt.part.WTPart:2200095",
       "AssemblyMode":
       {"Value": "separable"}
```

```
        ,
        "DefaultUnit" :
        {"Value": "ea"}
        ,
        "DefaultTraceCode":
        {"Value": "S"}
    }
   ]
}
```

Creating a Classified Part

This example shows you how to create a classified part. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/v3/ProdMgmt/Parts HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
    "ClfNodeInternalName": "Part",
    "ClfNodeDisplayName": "Part",
    "ClfNodeHierarchyDisplayName": "Part",
    "ClassificationAttributes": [
        {
            "InternalName": "xje136",
            "DisplayName": "Weight",
            "Value": "0.0",
            "DisplayValue": "0.0"
        },
        {
            "InternalName": "CB89D0417f",
            "DisplayName": "General Description",
            "Value": "0",
            "DisplayValue": "0"
        }
    ]
}
```

Retrieving Information About Classification Attributes

This example shows you how to retrieve information about classification attributes for the specified classification node. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/GetClassificationNodeInfo
(clfStructureNameSpace='com.ptc.csm.default_namespace',clfNodeInternalName='Part') HTTP/1.1
```

Response


```
{
    "@odata.context": "/Windchill/servlet/odata/ClfStructure/$metadata#ClassificationInfo",
    "ClfNodeInternalName": "Part",
    "ClfNodeDisplayName": "Part",
    "ClfNodeHierarchyDisplayName": "Part",
    "ClassificationAttributes": [
        {
            "InternalName": "xje136",
            "DisplayName": "Weight",
            "Value": "0.0",
            "DisplayValue": "0.0"
        },
        {
            "InternalName": "CB89D0417f",
            "DisplayName": "General Description",
            "Value": "0",
            "DisplayValue": "0"
        }
    ]
}
```


The response returned by the function can be specified as payload to create a classified part.

Creating a Document in a Different Organization

This example shows you how to create a document in a different organization. In Windchill, set the preference **Expose Organization** in **Preference Management** utility to `Yes`. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/DocMgmt/Documents HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Name": "TestDoc1",
"Description": "TestDoc1_Description",
"Title": "TestDoc1_Title",
   "Organization@odata.bind": "Organizations('OR:wt.inf.container.OrgContainer:373739)"
"Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:48788507')"
}
```

Creating a Document

This example shows you how to create a document. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/DocMgmt/Documents HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Name": "TestDoc1",
"Description": "TestDoc1_Description",
"Title": "TestDoc1_Title",
"Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:48788507')"
}
```

Creating Multiple Documents

This example shows you how to create multiple documents. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/DocMgmt/CreateDocuments HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "Documents": [
     { "Name": "test doc1", "Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:302725')" }
     ,
     { "Name": "test doc2", "Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:302725')" }
   ]
}
```

Checking Out a Document

This example shows you how to check out a document. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/DocMgmt/Documents HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"CheckOutNote" : "This is checkout note."
}
```

Updating a Document

This example shows you how to update a document. Use the following PATCH URI with the request body.

URI

```
PATCH /Windchill/servlet/odata/DocMgmt/Documents('VR:wt.doc.WTDocument:48796553') HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "Description": "TestDoc1_Description_Update",
   "CustomAttribute": "This is Test Attribute"
}
```

Updating Multiple Documents

This example shows you how to update multiple documents. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/DocMgmt/UpdateDocuments HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
    "Documents": [
       { "ID":"OR:wt.doc.WTDocument:2276131", "Description":"Updated description1",
         "Title":"Updated title1" }
       ,
       { "ID":"OR:wt.doc.WTDocument:2276126", "Title":"Updated title2" }
   ]
}
```

Uploading Content for a Document

This example shows you how to upload content for a document in the following cases:

- Using a local file
- Using URL data
- Using external data

Use the following POST URI with the request body.

Using a Local File

The content can be uploaded in the following stages:

- Stage1—POST URI

```
POST /Windchill/servlet/odata/DocMgmt/Documents('OR:wt.doc.WTDocument:48796581')/
    PTC.DocMgmt.uploadStage1Action HTTP/1.1
```

Stage 1—Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Stage1—Request Body

```
{
"noOfFiles":3
}
```

Stage1—Sample Output

```
{
 "@odata.context": "$metadata#CacheDescriptor",
 "value": [
  {
   "ID": null,
   "ReplicaUrl": "https://windchill.ptc.com/Windchill/servlet/WindchillGW/
    wt.fv.uploadtocache.DoUploadToCache_Server/doUploadToChache_Master?mk=
    wt.fv.uploadtocache.DoUploadToCache_Server&VaultId=150301&FolderId=150329&
    CheckSum=456186&sT=1507542170&sign=Ca4ouGGOZiopnqbd4mbUVg%3D%3D&site=
    https%3A%2F%2Fwindchill.ptc.com%2FWindchill%2Fservlet%2F
    WindchillGW&AUTH_CODE=HmacMD5&isProxy=true&delegate=
    wt.fv.uploadtocache.DefaultRestFormGeneratorDelegate",
   "MasterUrl": "https://windchill.ptc.com/Windchill/servlet/WindchillGW",
   "VaultId": 150301,
   "FolderId": 150329,
   "StreamIds": [
                76030,
                76032,
                76031
      ],
   "FileNames": [
                76030,
                76032,
                76031
           ]
       }
]
```

```
}
```

- Stage2—The HTTP request for Stage2 must be constructed from `ReplicaUrl` attribute which is retrieved from Stage1.

  Stage2—POST URI

```
https://windchill.ptc.com/Windchill/servlet/WindchillGW/
  wt.fv.uploadtocache.DoUploadToCache_Server/doUploadToChache_Master?
  mk=wt.fv.uploadtocache.DoUploadToCache_Server&VaultId=150301
  &FolderId=150329&CheckSum=456186&sT=1507542170&sign=
  Ca4ouGGOZiopnqbd4mbUVg%3D%3D&site=https%3A%2F%2F
  windchill.ptc.com%2FWindchill%2Fservlet%2F
WindchillGW&AUTH_CODE=HmacMD5&isProxy=true&delegate=
wt.fv.uploadtocache.DefaultRestFormGeneratorDelegate
```

  Stage 2—Request Headers

```
Content-Type: multipart/form-data; boundary=----------------------------boundary
```

  Stage2—Request Body

```
----------------------------boundary
Content-Disposition: form-data; name="Master_URL"
https://windchill.ptc.com/Windchill/servlet/WindchillGW
----------------------------boundary
Content-Disposition: form-data; name="CacheDescriptor_array"
76030: 76030: 76030; 76031: 76031: 76031; 76032: 76032: 76032;
----------------------------boundary
Content-Disposition: form-data; name="76030"; filename="TestFile1.txt"
is content of test file 1.
----------------------------boundary
Content-Disposition: form-data; name="76031"; filename="TestFile3.txt"
is content of test file 3.
----------------------------boundary
Content-Disposition: form-data; name="76032"; filename="TestFile2.txt"
is content of test file 2.
----------------------------boundary
```

Note

The `CacheDescriptor_array` contains the following information `<streamid>:<filename>:<contentid>:<filesize>` where,

- `streamid`—Specifies the unique content ID from the Stage1 response.
- `filename`—Specifies the name of the file from the Stage1 response.
- `contentid`—Same as `streamid`.
- `filesize`—Specifies size of the file to be uploaded in bytes (Optional).

The response from Stage2 contains information about the `streamId`, size of the file created, and encoded `CachedContentDescriptor`, which is used in Stage3 for uploading content to the document.

Stage2—Sample Output

```
{
"contentInfos": [
{
"streamId": 76030,
"fileSize": 2,
"encodedInfo": "76035%3A2%3A150329%3A76035"
},
{
"streamId": 76031,
"fileSize": 2,
"encodedInfo": "76034%3A2%3A150329%3A76034"
},
{
"streamId": 76032,
"fileSize": 2,
"encodedInfo": "76033%3A2%3A150329%3A76033"
}
]
}
```

- Stage3—POST URI

```
POST /Windchill/servlet/odata/DocMgmt/Documents('OR:wt.doc.WTDocument:48796581')/
     PTC.DocMgmt.uploadStage3Action HTTP/1.1
```

Stage 3—Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Stage3—Request body

```
{
"contentInfo" : [
{
"StreamId" :76030,
"EncodedInfo" : "76033%3A2%3A150329%3A76033",
"FileName" : "DesignSpec.doc",
"PrimaryContent" : true,
"MimeType" : "application/vnd.openxmlformats-officedocument.wordprocessingml.
document",
"FileSize" : 2
},
{
"StreamId" :76031,
"EncodedInfo" : "76035%3A2%3A150329%3A76035",
"FileName" : "ReferenceDoc1.doc",
"PrimaryContent" : false,
"MimeType" : "application/vnd.openxmlformats-officedocument.wordprocessingml.
document",
"FileSize" : 2
},
{
"StreamId" :76032,
"EncodedInfo" : "76034%3A2%3A150329%3A76034",
"FileName" : "ReferenceDoc2.doc",
"PrimaryContent" : false,
"MimeType" : "application/vnd.openxmlformats-officedocument.wordprocessingml.
document",
"FileSize" : 2
}
]
}
```

Using a URL Data

To create or update primary content from URL data, use the following PUT URI with the request body.

URI

```
PUT /Windchill/servlet/odata/DocMgmt/Documents('OR:wt.doc.WTDocument:2626068')/
    PrimaryContent HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "UrlLocation" :"https://www.ptc.com",
   "DisplayName" : "Test_PrimaryContent"
}
```

Using External Storage

To create or update the primary content from external storage, use the following PUT URI with the request body.

URI

```
PUT /Windchill/servlet/odata/DocMgmt/Documents('OR:wt.doc.WTDocument:2626068')/
   PrimaryContent HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "ExternalLocation" :"TestExternalLocation",
   "DisplayName" : "TestExternalLocation_DisplayName"
}
```

To create new attachments, use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/DocMgmt/Documents('OR:wt.doc.WTDocument:2626099')/
   Attachments HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "ExternalLocation" :"TestExternalLocation",
   "DisplayName" : "TestExternalLocation"
}
```

To update existing attachments, use the following PUT URI with the request body.

URI

```
PUT /Windchill/servlet/odata/DocMgmt/Documents('OR:wt.doc.WTDocument:2626099')/
     Attachments('OR:wt.content.ExternalStoredData:2626811') HTTP/1.1
```

Request Body

```
{
   "ExternalLocation" :"TestExternalLocation_Update",
   "DisplayName" : "TestExternalLocation_Update"
}
```

Examples for the PTC Data Administration Domain

Create a Folder and Subfolder

This example shows you how to create a folder and subfolder. Use the following POST URI with the request body.

URI to Create a Folder

```
POST /Windchill/Servlet/odata/DataAdmin/Containers(<oid>)/Folders HTTP/1.1
```

URI to Create a Subfolder

```
POST /Windchill/Servlet/odata/DataAdmin/Containers(<oid>)/
Folders(<oid>)/Folders HTTP/1.1
```

You can pass the following information in the request header and body for both the URIs.

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Name": "Demo",
“Description”: “Folder for CAD parts”
}
```

Update a Folder

This example shows you how to update a folder. Use the following PATCH URI with the request body.

URI

```
PATCH /Windchill/Servlet/odata/DataAdmin/Containers(<oid>)/
Folders(<oid>) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"Name": "Demo_Updated",
“Description”: “<Description_Update>”
```

```
}
```

Delete a Folder

This example shows you how to delete a folder. Use the following DELETE URI with the request body.

URI

```
DELETE /Windchill/Servlet/odata/DataAdmin/Containers(<oid>)/
Folders(<oid>) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Examples for the PTC Principal Management Domain

Retrieving License Groups for a User

This example shows you how to retrieve license groups for a user. Specify the `OID` for the user. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PrincipalMgmt/Users('<oid>')/LicenseGroups HTTP/1.1
```

Retrieving License Groups for a User with Expanded Navigation

This example shows you how to retrieve license groups for a user with expanded navigation. Specify the `OID` for the user. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PrincipalMgmt/Users('<oid>')?$expand=LicenseGroups HTTP/1.1
```

Examples for the PTC Parts List Management Domain

Retrieving Parts List

This example shows you how to retrieve parts list. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists HTTP/1.1
```

Retrieving Information for a Specific Part List

This example shows you how to retrieve a specific part list. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:197855') HTTP/1.1
```

Retrieving Parts List Items for a Specific Part List

This example shows you how to retrieve parts list items for a specific part list. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:197855')/Uses HTTP/1.1
```

Retrieving a Specific Part List Item

This example shows you how to retrieve a specific part list item. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:197855')/
Uses('OR:com.ptc.arbortext.windchill.partlist.PartListItem:240518') HTTP/1.1
```

Retrieving Parts from a Part List with Expanded Navigation

This example shows you how to retrieve parts from a part list item with expanded navigation. The response returns information about a part list item along with information about parts that are related to the part list item. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:197855')/Uses?$expand=Uses HTTP/1.1
```

Retrieving EPMDocuments from a Part List with Expanded Navigation

This example shows you how to retrieve EPMDocuments from a part list item with expanded navigation. The response returns information about illustrations along with information about EPMDocuments that are related to the illustrations. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:240515')/
```

```
DescribedBy('OR:com.ptc.arbortext.windchill.partlist.PartListToEPMDocumentLink:240527')?$
expand=DescribedBy HTTP/1.1
```

Retrieving Illustrations for a Parts List

This example shows you how to retrieve illustration for a parts list. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:197856')/DescribedBy HTTP/1.1
```

Retrieving a Specific Illustration

This example shows you how to retrieve a specific illustration. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartLists
('OR:com.ptc.arbortext.windchill.partlist.PartList:240515')/
DescribedBy('OR:com.ptc.arbortext.windchill.partlist.PartListToEPMDocumentLink:240527') HTTP/1.1
```

Retrieving a Specific Substitute Part

This example shows you how to retrieve a specific substitute part from a part list item. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartListItems
('OR:com.ptc.arbortext.windchill.partlist.PartListItem:240521')/
Substitutes('OR:com.ptc.arbortext.windchill.partlist.PartListItemSubstituteLink:243635') HTTP/1.1
```

Retrieving a Specific Supplementary Part

This example shows you how to retrieve a specific supplementary part from a part list item. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/PartListMgmt/PartListItems
('OR:com.ptc.arbortext.windchill.partlist.PartListItem:240521')/
Supplements('OR:com.ptc.arbortext.windchill.partlist.SupplementaryReplacementLink:243640') HTTP/1.1
```

Examples for the PTC Service Information Management Domain

Retrieving Information Structures

This example shows you how to retrieve all the information structures. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ServiceInfoMgmt/InformationStructures HTTP/1.1
```

To retrieve a specific information structure, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ServiceInfoMgmt/InformationStructures
('OR:wt.part.WTPart:258669') HTTP/1.1
```

Retrieving Publication Structures

This example shows you how to retrieve all the publication structures. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ServiceInfoMgmt/PublicationStructures HTTP/1.1
```

To retrieve a specific publication structure, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ServiceInfoMgmt/PublicationStructures
('OR:wt.part.WTPart:258623') HTTP/1.1
```

Retrieving Publication Structures for Authoring Language French

This example shows you how to retrieve all the publication structures, where the authoring language is set to French. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ServiceInfoMgmt/PublicationStructures
?$filter=AuthoringLanguage/Value eq 'fr' HTTP/1.1
```

Retrieving and Expanding the Contents of an Information Structure

This example shows you how to retrieve and expand the contents of an information structure. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/ServiceInfoMgmt/InformationStructures('OR:wt.part.WTPart:258669')/
```

```
PTC.ServiceInfoMgmt.GetStructure?$expand=Children($levels=max) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
                "NavigationCriteriaId":"OR:wt.filter.NavigationCriteria:270048"
}
```

You can choose not to specify the navigation criteria. In this case, the request body is empty, that is, `{}`. The default navigation criteria is used.

Retrieving and Expanding the Contents of a Publication Structure

This example shows you how to retrieve and expand the contents of a publication structure. The retrieved contents are sorted by line number. Use the following POST request, with the `ServiceObjectUsesLineNumber` property.

URI

```
POST /Windchill/servlet/odata/ServiceInfoMgmt/PublicationStructures('OR:wt.part.WTPart:258646')/
PTC.ServiceInfoMgmt.GetStructure?$expand=Children($levels=max ;$orderby=ServiceObjectUsesLineNumber)
HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
                "NavigationCriteriaId":"OR:wt.filter.NavigationCriteria:270048"
}
```

You can choose not to specify the navigation criteria. In this case, the request body is empty, that is, `{}`. The default navigation criteria is used.

Example for the PTC Info*Engine System Domain

Invoking an Info*Engine Task

This example shows you how to invoke an Info*Engine task.

Consider a task:

`wt/federation/delegates/windchill/QueryObjects.xml`
The task has two inputs: the Windchill *type* and a *where* clause for querying objects.

Use the following POST URI to invoke the task.

URI

```
POST /Windchill/servlet/odata/IE/InvokeIETask
```

Request Body

```
{
    "Task": "wt/federation/delegates/windchill/QueryObjects.xml",
    "Params": [
        {
            "Name": "type",
            "Value": "wt.part.WTPart"
        },
        {
            "Name": "where",
            "Value": "name=GOLF_CART"
        }
    ]
}
```

The output of this task is a JSON representation of groups and elements contained in the output group of the task.

Examples for the PTC Manufacturing Process Management Domain

Retrieving Operations For a Process Plan Using System Default NCs

This example shows you how to retrieve the first-level operations for a process plan. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/MfgProcMgmt/ProcessPlans('OR:com.ptc.windchill.mpml.processplan.
MPMProcessPlan:44148884')/PTC.MfgProcMgmt.Operations(processPlanNavigationCriteriaId='',
relatedAssemblyNavigationCriteriaId='') HTTP/1.1
```

Request Headers

```
Content-Type: application/json
```

Retrieving Illustration-Related Contents from an Operation

This example shows you how to retrieve the illustration-related contents associated with an operation. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/MfgProcMgmt/Operations
('OR:com.ptc.windchill.mpml.processplan.operation.MPMOperation:55667788')/
PTC.MfgProcMgmt.DownloadUrls() HTTP/1.1
```

Request Headers

```
Content-Type: application/json
```

Retrieving Document-Related Contents from a Sequence

This example shows you how to retrieve the primary contents of a document associated with a sequence. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/MfgProcMgmt/Sequences
('OR:com.ptc.windchill.mpml.processplan.sequence.MPMSequence:11223344')/
DocumentDescribeLinks?$expand=DescribedBy($expand=PrimaryContent) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
```

Reading the Bill of Process (BOP)

This example shows you how to read the bill of process (BOP) for a process plan. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/MfgProcMgmt/ProcessPlans
('OR:com.ptc.windchill.mpml.processplan.MPMProcessPlan:44148884')/
PTC.MfgProcMgmt.GetBOP?$expand=Components($expand=OperationHolder($select=Name,Number),
OperationHolderUsageLink;$levels=max) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
"processPlanNavigationCriteriaId" : "OR:wt.filter.NavigationCriteria:48796407",
"relatedAssemblyNavigationCriteriaId" : "OR:wt.filter.NavigationCriteria:48796408"
}
```

Examples for the PTC Change Management Domain

Retrieving Problem Reports

This example shows you how to retrieve problem reports. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ProblemReports HTTP/1.1
```

To retrieve a specific problem report, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ProblemReports
('OR:wt.change2.WTChangeIssue:233708') HTTP/1.1
```

Retrieving Variances

This example shows you how to retrieve variances. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/Variances HTTP/1.1
```

To retrieve a specific variance, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/Variances
('OR:wt.change2.WTVariance:233682') HTTP/1.1
```

Retrieving Variances Along with Variance Owners Information

This example shows you how to retrieve information about variance owners along with variances. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/Variances?$expand=VarianceOwners HTTP/1.1
```

To retrieve a specific variance, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/Variances
('OR:wt.change2.WTVariance:233682')?$expand=VarianceOwners HTTP/1.1
```

Retrieving Change Requests

This example shows you how to retrieve change requests. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests HTTP/1.1
```

To retrieve a specific change request, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:229667') HTTP/1.1
```

Retrieving Change Notices

This example shows you how to retrieve change notices. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeNotices HTTP/1.1
```

To retrieve a specific change notice, use the following GET request:

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/
ChangeNotices('OR:wt.change2.WTChangeOrder2:234109') HTTP/1.1
```

Retrieving the Process Links for Change Objects

This example shows you how to retrieve the process links for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests?$
expand=ProcessLinks HTTP/1.1
```

To retrieve the process links for a specific change object, use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:250651')?$expand=ProcessLinks HTTP/1.1
```

Retrieving Process Links for a Specific Change Object

This example shows you how to retrieve process link for a specific change object. Use the following GET request.

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/
ChangeRequests('OR:wt.change2.WTChangeRequest2:251664')/ProcessLinks HTTP/1.1
```

Retrieving the Process Objects for Change Objects

This example shows you how to retrieve the process objects for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests?$
expand=ProcessObjects HTTP/1.1
```

Retrieving Process Objects for a Specific Change Object

This example shows you how to retrieve process objects for a specific change object. Use the following GET request.

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/
ChangeRequests('OR:wt.change2.WTChangeRequest2:251664')/ProcessObjects HTTP/1.1
```

Retrieving the Reference Objects for Change Objects

This example shows you how to retrieve the reference objects for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests?$
```

```
expand=ReferenceObjects HTTP/1.1
```

Retrieving Reference Objects for a Specific Change Object

This example shows you how to retrieve reference objects for a specific change object. Use the following GET request.

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/
ChangeRequests('OR:wt.change2.WTChangeRequest2:251664')/ReferenceObjects HTTP/1.1
```

Retrieving the Reference Links for Change Objects

This example shows you how to retrieve the reference links for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests?$
expand=ReferenceLinks HTTP/1.1
```

Retrieving Reference Links for a Specific Change Object

This example shows you how to retrieve reference link for a specific change object. Use the following GET request.

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeMgmt/
ChangeRequests('OR:wt.change2.WTChangeRequest2:251664')/ReferenceLinks HTTP/1.1
```

Retrieving the Affected Links

This example shows you how to the retrieve `AffectedLinks` for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:250300')?$expand=AffectsLinks HTTP/1.1
```

Retrieving the AffectedByLinks

This example shows you how to retrieve the `AffectedByLinks` links for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.part.WTPart:250486')?$expand=AffectedByLinks HTTP/1.1
```

Retrieving the Affected Objects

This example shows you how to retrieve the `AffectedObjects` for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests?$expand=AffectedObjects HTTP/1.1
```

To retrieve the `AffectedObjects` for a specific change object, use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:250300')?$expand=AffectedObjects HTTP/1.1
```

Retrieving the AffectedBy Objects

This example shows you how to retrieve the `AffectedByObjects` for change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/Changeables?$
expand=AffectedByObjects HTTP/1.1
```

To retrieve the `AffectedByObjects` for a specific change object, use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:250687')?$expand=AffectedByObjects HTTP/1.1
```

Retrieving the Affected Links and Affected Objects

This example shows you how to retrieve the `AffectedLinks` for change objects, and further retrieve the `AffectedObjects`. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:250300')?$expand=AffectsLinks($expand=AffectedObjects) HTTP/1.1
```

Retrieving and Downloading the Attachments Associated with Change Objects

This example shows you how to retrieve the list of attachments associated with the change objects. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:229667')/Attachments HTTP/1.1
```

To view and download the attachment, use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeRequests
('OR:wt.change2.WTChangeRequest2:229667')/Attachments
```

```
('OR:wt.content.ApplicationData:278667')/$value HTTP/1.1
```

Retrieving the Resulting Links

This example shows you how to the retrieve resulting links for change notices. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeNotices?$expand=ResultingLinks HTTP/1.1
```

To get details of resulting links and change tasks (`ResultedByObjects`) , use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeNotices('OR:wt.change2.WTChangeOrder2:264082')
?$expand=ResultingLinks($expand=ResultedByObjects) HTTP/1.1
```

Retrieving the Resulting Objects

This example shows you how to the retrieve resulting objects for change notices. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeNotices?$expand=ResultingObjects HTTP/1.1
```

Retrieving the Unincorporated Links

This example shows you how to the retrieve unincorporated links for change notices. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ChangeMgmt/ChangeNotices?$expand=UnincorporatedLinks HTTP/1.1
```

Examples for the PTC Classification Structure Domain

Query Classification Nodes

Retrieving the First Child Node of a Root Node

This example shows you how to retrieve the first child node of a root node. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/ClfNodes HTTP/1.1
```

Retrieving a Specific Classification Node

This example shows you how to retrieve a specific classification node. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/ClfNodes('SPRING') HTTP/1.1
GET /Windchill/servlet/odata/ClfStructure/ClfNodes('ElectronicParts') HTTP/1.1
```

Retrieving the Parent Node of a Classification Node

This example shows you how to retrieve the parent node of the specified classification node. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/ClfNodes('SPRING')/ParentNode HTTP/1.1
```

Retrieving the Child Nodes of a Classification Node

This example shows you how to retrieve all the child nodes of the specified classification node. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/ClfNodes('SPRING')/ChildNodes HTTP/1.1
```

Query Classified Objects

Retrieving Classified Objects Associated with a Classification Node

This example shows you how to retrieve classified objects, which are associated with a classification node. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/ClfStructure/ClfNodes('SPRING')/
ClassifiedObjects HTTP/1.1
```

Retrieving a Specific Classified Object

This example shows you how to retrieve a specific classified object. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/ClfNodes('SPRING')/
ClassifiedObjects('OR:wt.part.WTPart:12345') HTTP/1.11
```

Retrieving Legal or Enumeration values for a Classification Attribute

This example shows you how to retrieve legal or enumeration values for the specified classification attribute. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/GetEnumTypeConstraintOnClfAttributes
(nodeInternalName='SPRING',clfStructureNameSpace='com.ptc.csm.default_clf_namespace',
attributeInternalName='color') HTTP/1.1
```

Retrieving Binding Attributes for Classified Object

This example shows you how to retrieve information about the classification binding attribute for a classified object. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ClfStructure/
GetClfBindingInfo(oid='OR:wt.part.WTPart:12345',
clfStructureNameSpace='com.ptc.csm.default_clf_namespace') HTTP/1.1
```

Examples for the PTC Saved Search Domain

Retrieving Saved Searches

This example shows you how to retrieve saved searches. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/SavedSearch/SavedQueries HTTP/1.1
```

To retrieve a specific saved search, use the following GET request:

URI

```
GET /Windchill/servlet/odata/SavedSearch/
SavedQueries('OR:wt.query.SavedQuery:322032') HTTP/1.1
```

Executing a Saved Search

This example shows you how to execute a saved search. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/SavedSearch/
SavedQueries('OR:wt.query.SavedQuery:313313')/
PTC.SavedSearch.ExecuteSavedSearch(Keyword='') HTTP/1.1
```

Retrieving Object Types for a Saved Searches

This example shows you how to retrieve the object types for a saved search. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/SavedSearch/SavedQueries
('OR:wt.query.SavedQuery:311158')/
PTC.SavedSearch.GetSelectedTypesFromSavedSearch() HTTP/1.1
```

Examples for the PTC Visualization Domain

Retrieving a Representation

This example shows you how to download a representation. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/Visualization/Representations
('OR:wt.viewmarkup.DerivedImage:786687') HTTP/1.1
```

You can use the function `GetPVZ()` to download a representation. For this representation, pass the parameter value as `true` for *IncludeAnnotations*. Pass the fidelity value as `Low Fidelity` for the parameter *Fidelity*.

URI

```
GET /Windchill/servlet/odata/Visualization/Representations
('OR:wt.viewmarkup.DerivedImage:786687')/
Visualization.GetPVZ(IncludeAnnotations=true,Fidelity='Low Fidelity') HTTP/1.1
```

Retrieving Fidelity Names

This example shows you how to retrieve the fidelity values associated with a representation. The function `GetFidelities()` retrieves the fidelity values as `PTC.EnumType`, in a value-display format. Use the following GET request.

URI

```
GET  /Windchill/servlet/odata/Visualization/Representations
('OR:wt.viewmarkup.MultiFidelityDerivedImage:786687')/Visualization.GetFidelities() HTTP/1.1
```

Examples for the PTC Audit Domain

Retrieving Audits

This example shows you how to retrieve audits. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/Audit/Audits HTTP/1.1
```

Retrieving Information for a Specific Audit

This example shows you how to retrieve information for a specific audit. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/Audit/Audits
('OR:com.ptc.qualitymanagement.audit.WTChangeAudit:267253')/AuditDetails/ HTTP/1.1
```

Updating Audit Score for a Specific Audit Detail

This example shows you how to update the audit score for a specific audit detail. Use the following GET request.

URI

```
PATCH /Windchill/servlet/odata/Audit/Audits
('OR:com.ptc.qualitymanagement.audit.WTChangeAudit:267253')
/AuditDetails('OR:com.ptc.qualitymanagement.audit.AuditDetail:267264')/ HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
    “OnSiteAuditScore”:82
}
```

Examples for the PTC Product Platform Management Domain

Retrieving Options

This example shows you how to retrieve options. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/ProdPlatformMgmt/Options HTTP/1.1
```

Retrieving Option Groups for All Options

This example shows you how to retrieve option groups for all options. Use the following GET request

URI

```
GET /Windchill/servlet/odata/ProdPlatformMgmt/Options?$expand=OptionGroup HTTP/1.1
```

Retrieving the Option Group for a Specific Option

This example shows you how to retrieve the option group for a specific option. Use the following GET request

URI

```
GET /Windchill/servlet/odata/ProdPlatformMgmt/Options
('OR:com.ptc.windchill.option.model.Option:139849')/OptionGroup HTTP/1.1
```

Retrieving a Specific Option With Option Group

This example shows you how to retrieve a specific option with its option group. Use the following GET request

URI

```
GET /Windchill/servlet/odata/PProdPlatformMgmt/Options
('OR:com.ptc.windchill.option.model.Option:139849')?$expand=OptionGroup HTTP/1.1
```

Examples for the PTC CAD Document Management Domain

Retrieving a Specific CAD Document

This example shows you how to retrieve a specific CAD document. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments
('OR%3Awt.epm.EPMDocument%3A167183') HTTP/1.1
```

Retrieving a Specific CAD Document with Expanded Navigation

This example shows you how to retrieve a specific CAD document with expanded navigation. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments
('OR%3Awt.epm.EPMDocument%3A167183')?$expand=Uses HTTP/1.1
```

Querying the CAD Document Using a Filter

This example shows you how to query a specific CAD document using a filter based on document name. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments?$filter=FileName eq 'ABC.prt' HTTP/1.1
```

Retrieving Related Parts Information for a Specific CAD Document

This example shows you how to retrieve related parts information for a specific CAD document. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments('OR%3Awt.epm.EPMDocument%3A167183')
?$expand=PartAssociations($expand=RelatedParts) HTTP/1.1
```

Note

The related parts information is not returned with this navigation in the following cases:

- When a CAD document contains model items that are related to a Part
- When a CAD document has a custom association to a Part
- When a CAD document drawing has a calculated association to a Part.

Retrieving References Information for a Specific CAD Document

This example shows you how to retrieve references information for a specific CAD document. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments
('OR%3Awt.epm.EPMDocument%3A167183')?$expand=References HTTP/1.1
```

Retrieving Source Information for a Specific Image CAD Document

This example shows you how to retrieve the source CAD document for a specific image CAD document. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments
('OR%3Awt.epm.EPMDocument%3A167183')?$expand=DerivedSources($expand=SourceCADDocuments) HTTP/1.1
```

Retrieving the CAD Structure using BOMMembersOnly Parameter

This example shows you how to retrieve a CAD structure using `BOMMembersOnly` parameter. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/v1/CADDocumentMgmt/CADDocuments('OR%3Awt.epm.EPMDocument%3A167183')/
PTC.CADDocumentMgmt.GetStructure?$expand=Components($levels=max) HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
```

```
    "BOMMembersOnly" : true
}
```

Examples for the PTC Event Management Domain

Examples for the PTC RegulatoryMaster Domain

Retrieving Regulatory Submissions

This example shows how to retrieve regulatory submissions. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/RegMstr HTTP/1.1
```

To retrieve a specific regulatory submission, use the following GET request.

URI

```
GET /Windchill/servlet/odata/RegMstr('OR:com.ptc.qualitymanagement.
regmstr.RegulatorySubmission:237897') HTTP/1.1
```

Creating a Regulatory Submission

This example shows how to create a regulatory submission. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/RegMstr/RegulatorySubmissions HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "@odata.type": "PTC.RegMstr.org.rnd.SampleInternal",
   "SubmittedTo@odata.bind": "Places{{PLACE_ID}}",
   "Context@odata.bind": "Containers('Container_ID')",
   "Subject@odata.bind": "Subjects{{wtPART_ID}}",
   "Name": "Test"
}
```

Subscribing to an Event of a Windchill Object Instance

This example shows you how to subscribe to an event that occurs on the specified Windchill object instance. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/EventMgmt/EventSubscriptions HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
    "Name": "TestSubscriptionForDoc",
    "CallbackURL": "https://windchill.ptc.com/Windchill",
    "SubscribedEvent@odata.bind": "Events(' CHANGE_LIFECYCLE_STATE')",
    "LifeCycleState":
   {"Value": "RELEASED"},
   "SubscribedOnEntity@odata.bind": "WindchillEntities('OR:wt.doc.WTDocument:4326293')",
   "SubscribeAllVersions": true,
   "@odata.type":"PTC.EventMgmt.EntityEventSubscription"
}
```

Subscribing to an Event of a Windchill Object Type in the Specified Container

This example shows you how to subscribe to an event that occurs on the specified type of Windchill object in the specified container. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/EventMgmt/EventSubscriptions HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
    "Name": "TestContainerSubscription",
    "CallbackURL": "https://windchill.ptc.com/Windchill",
    "SubscribedEvent@odata.bind": "Events('EDIT_IDENTITY')",
    "SubscribedOnEntityType": "PTC.DocMgmt.Document",
    "ExpirationDate": "2018-12-20T11:30:00Z",
    "SubscribedOnContext@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:79638')",
    "@odata.type":"PTC.EventMgmt.EntityTypeInContainerEventSubscription"
}
```

Subscribing to an Event of a Windchill Object Type in the Specified Folder

This example shows you how to subscribe to an event that occurs on the type of specified Windchill object in the specified folder. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/EventMgmt/EventSubscriptions HTTP/1.1
```

Request Headers

```
Content-Type: application/json
```

```
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
    "Name": "TestFolderSubscription",
    "CallbackURL": "https://windchill.ptc.com/Windchill",
    "SubscribedOnEntityType": "PTC.DocMgmt.Document",
    "ExpirationDate": "2018-12-20T11:30:00Z",
    "SubscribedEvent@odata.bind": "Events(' EDIT_ATTRIBUTES')",
    "SubscribedOnFolder@odata.bind": "Folders('OR:wt.folder.SubFolder:5012381')",
    "@odata.type":"PTC.EventMgmt.EntityTypeInFolderEventSubscription" }
```

Deleting a Subscription

This example shows you how to delete a subscription. Use the following DELETE request.

URI

```
DELETE /Windchill/servlet/odata/EventMgmt/EventSubscriptions
('OR:wt.notify.NotificationSubscription:5012541') HTTP/1.1
```

Examples for the PTC Supplier Management Domain

Retrieving Sourcing Contexts

This example shows you how to retrieve all the sourcing contexts. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/SupplierMgmt/SourcingContexts HTTP/1.1
```

Retrieving Supplier, Manufacturer, and Vendor Parts

This example shows you how to retrieve supplier, manufacturer, and vendor parts. Use the following GET requests.

URI for Supplier Parts

```
GET /Windchill/servlet/odata/ProdMgmt/SupplierParts HTTP/1.1
```

URI for Manufacturer Parts

```
GET /Windchill/servlet/odata/ProdMgmt/ManufacturerParts HTTP/1.1
```

URI for Vendor Parts

```
GET /Windchill/servlet/odata/ProdMgmt/VendorParts HTTP/1.1
```

Retrieving AXLEntry for Parts

This example shows you how to retrieve AXLEntry for parts. Use the following GET requests.

URI for Retrieving AXLEntry

```
GET /Windchill/servlet/odata/ProdMgmt/Parts?$expand=AXLEntries HTTP/1.1
```

URI for Retrieving AXLEntry for the Specified Part

```
GET /Windchill/servlet/odata/ProdMgmt/Parts(‘<Part_ID>’)/AXLEntries HTTP/1.1
```

URI for Retrieving a Souring Context for a Specified AXLEntry

```
GET /Windchill/servlet/odata/ProdMgmt/Parts('<Part_ID>')/
AXLEntries('<AXLEntry_ID>')/SourcingContext HTTP/1.1
```

Filtering Parts Based on Sourcing Context and Sourcing Status Using Lambda Expression

This example shows you how to filter parts based on sourcing context and sourcing status using the lambda expression. Use the following GET requests.

URI for Filtering Parts with Specified Sourcing Context ID and Sourcing Status Value

```
GET /Windchill/servlet/odata/ProdMgmt/Parts?$filter=(OEMPartSourcingStatus/
any(d:d/SourcingStatus/Value eq 'preferred' and
d/SourcingContext/SourcingContextId eq 'com.ptc.windchill.suma.axl.AXLContext:204742') HTTP/1.1
```

URI for Filtering Parts with Combinations of Sourcing Status and Contexts

```
GET /Windchill/servlet/odata/ProdMgmt/ProdMgmt/Parts?$filter=startswith(Name,'OEM') and
(OEMPartSourcingStatus/any(d:d/SourcingStatus/Value eq 'preferred' and
d/SourcingContext/SourcingContextId eq 'com.ptc.windchill.suma.axl.AXLContext:204742')
or OEMPartSourcingStatus/any(d:d/SourcingStatus/Value eq 'do_not_use'
and d/SourcingContext/SourcingContextId eq 'com.ptc.windchill.suma.axl.AXLContext:204742')) HTTP/1.1
```

Examples for the PTC Regulatory Master Domain

Retrieving Regulatory Submissions

This example shows how to retrieve regulatory submissions. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/RegMstr HTTP/1.1
```

To retrieve a specific regulatory submission, use the following GET request.

URI

```
GET /Windchill/servlet/odata/RegMstr('OR:com.ptc.qualitymanagement.
regmstr.RegulatorySubmission:237897') HTTP/1.1
```

Creating a Regulatory Submission

This example shows how to create a regulatory submission. Use the following POST URI with the request body.

URI

```
POST /Windchill/servlet/odata/RegMstr/RegulatorySubmissions HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body

```
{
   "@odata.type": "PTC.RegMstr.org.rnd.SampleInternal",
   "SubmittedTo@odata.bind": "Places{{PLACE_ID}}",
   "Context@odata.bind": "Containers('Container_ID')",
   "Subject@odata.bind": "Subjects{{wtPART_ID}}",
   "Name": "Test"
}
```

Examples for the PTC Workflow Domain

Retrieving Work Items

This example shows you how to retrieve all the work items. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/Workflow/WorkItems HTTP/1.1
```

To get a list of work items assigned to you, use the following GET request.

URI

```
GET /Windchill/Workflow/GetEnumTypeConstraint
(entityName='PTC.Workflow.WorkItem',propertyName='Status') HTTP/1.1
```

Querying the Work Items Using a Filter

This example shows you how to query work items using a filter based on activity and status. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/Workflow/WorkItems
?$expand=Activity&$filter=Activity/Name
eq 'Analyze Change Request' and Status/Display eq 'Potential' HTTP/1.1
```

Retrieving the Routing Options for a Work Item

This example shows you how to retrieve routing options for a specified work item. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/Workflow/WorkItems
('OR:wt.workflow.work.WorkItem:178380')/Activity/UserEventList HTTP/1.1
```

Retrieving the Subjects for Work Items

This example shows you how to retrieve subjects for work items. Use the following GET request.

URI

```
GET /Windchill/ servlet/odata/Workflow/WorkItems?$expand=Subject HTTP/1.1
```

Completing a Work Item

This example shows you how to complete a work item. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/Workflow/WorkItems
('OR:wt.workflow.work.WorkItem:{{<workitem_id>}}')/PTC.Workflow.CompleteWorkitem HTTP/1.1
```

You can pass the following information in the request header.

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Depending on what options you want to set, you can specify those options in the request body.

Request Body for Default Activity

```
{
"UserEventList":[],
"WorkitemComment":"Completing Workitem",
"VoteAction":"",
"AutomateFastTrack":false,
"Variables":[]
}
```

Request Body with Valid Routing Option

```
{
"UserEventList":[ "Reject"],
"WorkitemComment":"Completing Workitem",
"VoteAction":"",
"AutomateFastTrack":false,
"Variables":[]
}
```

Request Body with Valid Voting Option

```
{
"UserEventList":[],
"WorkitemComment":"Completing Workitem",
"VoteAction":"Approve",
"AutomateFastTrack":false,
"Variables":[]
}
```

Request Body with Variables

```
 {
"UserEventList":[],
```

```
"WorkitemComment":"Completing Workitem",
"VoteAction":"",
"AutomateFastTrack":false,
"Variables":[{
"Name":"act3_string",
"Value":"vxcvcvxcv"
},{
"Name":"act3_int",
"Value":"1234"
},{
"Name":"act3_boolean",
"Value":"false"
},{
"Name":"act3_date",
"Value":"01/05/2019"
}]
}
```

Saving a Work Item

This example shows you how to save a work item. Use the following POST request.

URI

```
POST /Windchill/servlet/odata/v1/Workflow/WorkItems
('OR:wt.workflow.work.WorkItem:{{<workitem_id>}}')/PTC.Workflow.SaveWorkitem HTTP/1.1
```

You can pass the following information in the request header.

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Depending on what options you want to set, you can specify those options in the request body.

Request Body for Default Activity

```
{
"UserEventList":[],
"WorkitemComment":"Saving Workitem",
"VoteAction":"",
"AutomateFastTrack":false,
"Variables":[]
}
```

Request Body with Valid Routing Option

```
{
"UserEventList":["Accept"],
"WorkitemComment":"Saving Workitem",
"VoteAction":"",
"AutomateFastTrack":false,
"Variables":[]
}
```

Request Body with Valid Voting Option

```
{
"UserEventList":[],
```

```
"WorkitemComment":"Saving Workitem",
"VoteAction":"Do not approve",
"AutomateFastTrack":false,
"Variables":[]
}
```

Request Body with Variables

```
{
"UserEventList":[],
"WorkitemComment":"Saving Workitem",
"VoteAction":"",
"AutomateFastTrack":false,
"Variables":[{
"Name":"act3_string",
"Value":"vxcvcvxcv"
},{
"Name":"act3_int",
"Value":"1234"
},{
"Name":"act3_boolean",
"Value":"false"
},{
"Name":"act3_date",
"Value":"01/05/2019"
}]
}
```

Retrieving the List of Valid Users to Reassign Work Items

This example shows you how to retrieve list of users to whom the specified work items can be reassigned. Use the following GET request.

URI

```
GET /Windchill/servlet/odata/v1/Workflow/GetWorkItemReassignUserList(WorkItems=@wi)?@wi=
["OR:wt.workflow.work.WorkItem:232279","OR:wt.workflow.work.WorkItem:232290",
"OR:wt.workflow.work.WorkItem:232297"] HTTP/1.1
```

`@wi` represents an alias, which can be replaced with any other name.

Reassigning Work Items to Another User

This example shows you how to reassign work items to another user. Use the following POST request.

URI

```
GET /Windchill/servlet/odata/Workflow/ReassignWorkItems HTTP/1.1
```

Request Headers

```
Content-Type: application/json
CSRF_NONCE: <Use the value from Fetch NONCE example>
```

Request Body for Default Activity

```
{ "WorkItems": [{"ID":"OR:wt.workflow.work.WorkItem:162155"}],
  "User": {"ID":"OR:wt.org.WTUser:11"},
  "Comment": "<Reassign_Comment_Optional>" }
```

Customizing Domains

Windchill REST Services enables you to customize domains provided by PTC. You can also add new domains. For customizing existing domains and creating new domains, new configuration files must be created in the custom configuration path `<Windchill>/codebase/rest/custom/domain`. The configurations from the custom configuration path are merged with the customizations in the PTC configuration path `<Windchill>/codebase/rest/ptc/domain`.

Note

Changes made in the PTC configuration path are not supported. The changes made here will be overwritten in the next update.

Extending Domains

To extend a PTC domain, mirror the domain folder structure from the PTC configuration path to the custom configuration path. Only the folder structure is mirrored. Copies of `.json` or `.js` files are not created. After the folder structure is mirrored, customizers decide what do they want to extend in the PTC domain, and create the required `.json` and `.js` files to extend domain definitions.

Windchill REST Services supports the following types of domain extensions:

- Adding type extensions of Windchill types to PTC Domains
- Adding custom properties to entities in PTC Domains
- Adding custom navigation between entities in PTC Domains
- Adding new functions to PTC Domains
- Adding new actions to PTC Domains

Adding Type Extensions of Windchill Types to PTC Domains

A PTC domain can be extended to add an OData entity that corresponds to a custom soft type created in Windchill. Customizers often create a custom soft type extension in Windchill to add a new behavior to Windchill. For example, consider the case where customizers have created a subclass of `WTPart`. A soft type `com.custom.PurchasePart` is created for `WTPart`. Further an additional string attribute called `SupplierName` on `PurchasePart` is also added.

To enable this soft type in Product Management domain, customizers first mirror the ProdMgmt domain folder structure in the custom configuration path. Then, create the `PurchasePart.json` file. Perform the following steps to enable a soft type:

1. In the custom configuration path, create the following folder structure for the Product Management domain at `<Windchill>/codebase/rest/custom/domain/`:
   - `ProdMgmt`
     - `v2`
       - `entity`
2. Create the `PurchaseParts.json` file at `<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity` and add the following content in the file:

```
{
   "name": "PurchasePart",
   “type”: “wcType”,
   “wcType”: “com.custom.PurchasePart”,
   “collectionName”: “PurchaseParts”,
   “includeInServiceDocument”: “false”,
   "parent": {
       “name”: "Part"
   },
   "attributes": [
       {
        "name": "SupplierName",
        "internalName": "SupplierName",
        "type": "String"
       }
   ]
}
```

After the configuration, when you visit the metadata URL for Product Management domain, the new entity `PurchasePart`, which is derived from the part entity is available. The `PurchasePart` entity also has the `SupplierName` property. Since the `PurchasePart` entity is now in the EDM, standard OData URLs can be used to access `PurchasePart`.

Adding Custom Properties to Entities in PTC Domains

A PTC domain can be extended to have custom properties which have been added to Windchill types by customizers. Customizers add new properties for Windchill types such as, `WTParts`, `WTDocuments`, and so on. `WTParts` and `WTDocuments` are available as Part and Document entities in the Product Management and Document Management domains respectively. You can add new

attributes as properties for these entities. To add `OwningBusinessUnit` and `DesignCost` attributes to the Part entity from the Product Management domain, the customizers mirror the ProdMgmt domain folder structure in the custom configuration path. Then, create the `PartsExt.json` file to add custom configuration. In the JSON file, under `extends` property, add the PTC domain entity which you want to extend. In the `attributes` property, add the new attributes. Perform the following steps to add custom properties to entities:

1. In the custom configuration path, create the following folder structure for the Product Management domain at `<Windchill>/codebase/rest/custom/domain/`:
   - `ProdMgmt`
     - `v2`
       - `entity`
2. Create the `PartsExt.json` file at `<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity` and add the following content in the file:

```
{
   "extends": "Parts",
   "attributes": [
       {
        "name": "OwningBusinessUnit",
        "internalName": "OwningBusinessUnit",
        "type": "String"
       },
     {
        "name": "DesignCost",
        "internalName": "DesignCost",
        "type": "Double"
       }
   ]
}
```

After the configuration, when you visit the metadata URL for Product Management domain, it shows the new properties `OwningBusinessUnit` and `DesignCost` on the Part entity for ProdMgmt domain. Since the Part entity has additional attributes, they can be used in standard OData URLs.

Adding Custom Navigation Between Entities in PTC Domains

A PTC domain can be extended to have new navigation between entities in a PTC domain. For example, the Product Management domain, does not provide any navigation between Part entities to show parts that are alternates of each other. To provide this navigation, customizers must extend the Product Management

domain. To add `Alternates` navigation between Part entities in the Product Management domain, the customizers mirror the ProdMgmt domain folder structure in the custom configuration path. Then create the `PartsExt.json` file to add custom configuration. In the JSON file, under `extends` property, add the PTC domain entity which you want to extend. In the `navigations` property, add the new navigation. Apart from providing the configurations in the `.json` file, customizers must also provide the programming logic to create the target entity set while navigating from the source to target entity. This is done in the `.js` file corresponding to the entity, in this case, `PartsExt.js` file. Perform the following steps to add custom navigation between entities:

1. In the custom configuration path, create the following folder structure for the Product Management domain at `<Windchill>/codebase/rest/custom/domain/`:
   - `ProdMgmt`
     - `v2`
       - `entity`
2. Create the `PartsExt.json` file at `<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity` and add the following content in the file:

```
{
   "extends": "Parts",
   "navigations": [
       {
        "name": "Alternates",
        "target": "Parts",
        "type": "Part",
               "isCollection": true,
               "containsTarget": true
       }
   ]
}
```

3. Create the `PartsExt.js` file at `<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity` and implement the following hooks:
   - `getRelatedEntityCollection`—The hook returns the following information:
     a. Gets the alternate part entities from the source entities.
     b. Puts the alternate part entities in an entity collection.

c. Returns the entity collection in a map.

- `isValidNavigation`—The hook returns the following information:
  a. Checks if the navigation being carried out is Alternates. If not, it returns null so that the framework can continue processing other navigations.
  b. Gets the source part.
  c. Navigates to the target part.
  d. Verifies that the target part is the same as specified in the input.
  e. Returns true or false to indicate the success of the validation.

Many of the hooks have been implemented in PTC provided domains. For code examples of hook implementations, you can see their implementations in any of PTC provided domains.

After the configuration, when you visit the metadata URL for Product Management domain, the `Alternates` navigation is available for the Part entity. You can navigate from a part to get its alternate parts.

Adding New Functions to PTC Domains

A PTC domain can be extended to add both bound and unbound OData functions. OData functions appear in the EDM of a domain. They are invoked with a GET request to the Odata URL of the function. For example, consider a case where you want to add a bound function to the Product Management domain that identifies costly parts within an entity set Parts. Perform the following steps to add a bound function:

1. In the custom configuration path, create the following folder structure for the Product Management domain at `<Windchill>/codebase/rest/custom/domain/`:
   - `ProdMgmt`
     - `v2`
       - `entity`
2. Create the `PartsExt.json` file at `<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity` and add the following content in the file:

```
{
   "extends": "Parts",
   "functions": [
       {
          "name": "GetCostlyParts",
           "description": "Return expensive parts",
           "isComposable": false,
             "parameters": [
```

```
                "name": "PartSet",
                "type": "Part",
                "isCollection": true,
                "isNullable": false
            ],
            "returnType": {
                "type": "Part",
                "isCollection": true
            }
        ]
}
```

3. Create the `PartsExt.js` file at `<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity` and implement the function. Ensure that the Part entity has a numeric property `DevelopmentCost`.

```
function function_GetCostlyParts(data, params) {
    var ArrayList = Java.type('java.util.ArrayList');
    var EntityCollection = Java.type('org.apache.olingo.commons.api.data.EntityCollection');
    var parts = data.getProcessor().readEntitySetData(data);
    var partEntityMap = data.getProcessor().toEntities(parts, data);
    var partEntities = new ArrayList(partEntityMap.values());

    var entityCollection = new EntityCollection();
    for(var i = 0; i < partEntities.size(); i++) {
        var partEntity = partEntities.get(i);
        var partCostProperty = partEntity.getProperty('DevelopmentCost');
        if(partCostProperty) {
            var partCost = partCostProperty.getValue();
            if(partCost && partCost > 0.10) {
                entityCollection.getEntities().add(partEntity);
            }
        }
    }

    return entityCollection;
}
```

After the configuration, when you visit the metadata URL for Product Management domain, the `GetCostlyParts` function is available for the Part entity. You can call the function on the Parts entity set and get a list of the costly parts.

Adding New Actions to PTC Domains

OData actions change the state of the entities and are called with a POST request. These are the basic differences between actions and functions.

In terms of definition, actions are similar to functions. However, there are some differences in definition between actions and functions:

- Actions are defined in the `actions` property of imports and entity JSON files.
- Actions are named with a prefix of `action_`.

Creating New Domains

Windchill REST Services enables you to create new domains. The new domains are created in the custom configuration path.

To create a new domain, perform the following steps:

1. Decide a domain identifier and the domain version. Create the domain folder `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>/<Domain Version>`
2. Create the `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>.json` file and provide values for domain metadata attributes.
3. Decide which other domains to import and set up the `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>/<Domain Version>/import.json` file.
4. Decide if the domain must have unbound actions or functions and set up the `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>/<Domain Version>/import.json` and `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>/<Domain Version>/import.js` files.
5. If `complexTypes` are required then set up the complex type JSON files at `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>/<Domain Version>/complexType`.
6. Configure entities and entity relations at `<Windchill>/codebase/rest/custom/domain/<Domain Identifier>/<Domain Version>/entity`.

After these files are setup, the domain is available at the REST root URL and can be accessed by OData URLs.

These are generic instructions to create a domain. You have to create and configure files depending on the entities of the domain. In this User's Guide, we have provided an example, that shows how to create a domain. The example helps you understand which files to create while configuring a domain.

Examples for Customizing Domains

Creating a New Domain

This example shows you how to create a new domain.

Consider an example, where a new domain Reporting must be created for building a reporting application. The Windchill types, `WTChangeIssue` and `Changeable2` must be exposed as `ProblemReport` and `ChangeableItem` entities respectively. Further, the `ReportedAgainst` relationship between `ProblemReport` and `ChangeItem` entities must also be exposed. The version `v1` must also be set up for the Reporting domain. For the reporting purpose, information can only be read from Windchill. The following properties of the two entities of the domain are exposed:

- ProblemReport
  - Number
  - Name
  - Occurrence date
  - Need date
  - Priority
  - Category
  - State
- ChangeableItem
  - Number
  - Name
  - Revision
  - State

To configure a domain for all the criteria mentioned in the example, perform the following steps:

1. Create the folder `<Windchill>/codebase/rest/custom/domain/Reporting`.
2. Create the file `<Windchill>/codebase/rest/custom/domain/Reporting.json` with the following content:

```
{
  "name": "Reporting",
  "id": "Reporting",
  "description": "Reporting Domain",
  "nameSpace": "Custom.Reporting",
  "containerName": "Windchill",
  "defaultVersion": "1"
```

```
}
```

3. Create the folder <Windchill>/codebase/rest/custom/domain/Reporting/v1.
4. Create the file <Windchill>/codebase/rest/custom/domain/Reporting/v1/import.json with the following content:

```
{
   "imports": [
       {"name": "PTC", "version": "1"}
   ]
    }
```

5. Create the folder <Windchill>/codebase/rest/custom/domain/Reporting/v1/entity.
6. Create the file <Windchill>/codebase/rest/custom/domain/Reporting/v1/entity/ChangeableItems.json with the following content:

```
{
  "name": "ChangeableItem",
  "collectionName": "ChangeableItems",
  "type": "wcType",
  "wcType": "wt.change2.Changeable2",
  "description": "Changeable Item",
  "operations": "READ",
  "attributes": [
    {"name": "Name", "internalName": "name", "type": "String"},
    {"name": "Number", "internalName": "number", "type": "String"}
  ],
  "inherits": [
    {"name": "lifecycleManaged"},
    {"name": "versioned"}
  ]
}
```

7. Create the file <Windchill>/codebase/rest/custom/domain/Reporting/v1/entity/ProblemReports.json with the following content:

```
{
 "name": "ProblemReport",
 "collectionName": "ProblemReports",
 "type": "wcType",
 "wcType": "wt.change2.WTChangeIssue",
 "description": "Problem Report",
 "operations": "READ",
 "attributes": [
```

```
    {"name": "Name", "internalName": "name", "type": "String"},
    {"name": "Number", "internalName": "number", "type": "String"},
    {"name": "Priority", "internalName": "theIssuePriority", "type": "String"},
    {"name": "Category", "internalName": "theCategory", "type": "String"},
    {"name": "OccurrenceDate", "internalName": "occurrenceDate", "type": "DateTimeOffset"},
    {"name": "NeedDate", "internalName": "needDate", "type": "DateTimeOffset"}
  ],
 "navigations": [
 {"name": "ReportedAgainst", "target": "ChangeableItems", "type": "ChangeableItem",
    "containsTarget": true, "isCollection": true}
 ],
  "inherits": [
    {"name": "lifecycleManaged"}
  ]
}
```

8. Create the file `<Windchill>/codebase/rest/custom/domain/Reporting/v1/entity/ProblemReports.js` with the following content:

```
function getRelatedEntityCollection(navProcessorData) {
  var HashMap = Java.type('java.util.HashMap');
  var ArrayList = Java.type('java.util.ArrayList');
  var WTArrayList = Java.type('wt.fc.collections.WTArrayList');
  var ChangeHelper2 = Java.type('wt.change2.ChangeHelper2');
  var targetName = navProcessorData.getTargetSetName();
  var map = new HashMap();
  var sourcePRs = new WTArrayList(navProcessorData.getSourceObjects());
  if("ReportedAgainst".equals(targetName)) {
    for(var i = 0; i < sourcePRs.size(); i++) {
      var sourcePR = sourcePRs.getPersistable(i);
      var reportedAgainstItems = ChangeHelper2.service.getChangeables(sourcePR, true);
      var list = new ArrayList();
      while(reportedAgainstItems.hasMoreElements()) {
        list.add(reportedAgainstItems.nextElement());
      }
      map.put(sourcePR, list);
    }
  }
  return map;
}
```

This creates a new domain called Reporting with all the entities and relationships described in the example. To test the domain, use the following URLs:

- To see the EDM for the Reporting domain, use the URL:

  ```
  https://<Windchill server>/Windchill/servlet/odata/Reporting/$metadata
  ```

- To see the list of `ProblemReports`, use the URL:

  ```
  https://<Windchill server>/Windchill/servlet/odata/Reporting/ProblemReports
  ```

- To see the list of `ProblemReport` with `ChangeableItems`, use the URL:

  ```
  https://<Windchill server>/Windchill/servlet/odata/Reporting/ProblemReports?$
  expand=ReportedAgainst
  ```

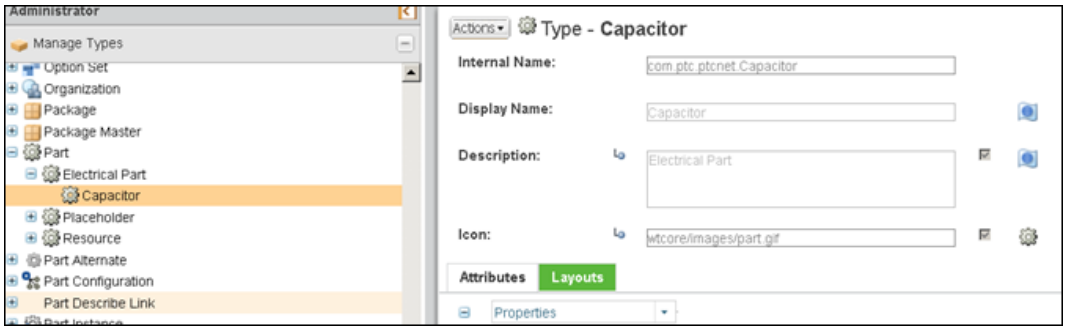
Extending Product Management Domain to Add A Soft Type

This example shows you how to extend the Product Management domain to add a soft type of an existing part. Consider a case where you want to create `WTPart` of soft type `Capacitor` which has its parent soft type as `Electrical Part`.

To extend the domain to add the soft type, create a custom configuration file `Capacitors.json` at:

`<Windchill>/codebase/rest/custom/domain/ProdMgmt/v2/entity`

`wcType` property must have the same **Internal Name** as defined for the soft type in **Type Management**.




```
{
    "name": "Capacitor",
    "collectionName": "Capacitors",
    "wcType": "com.ptc.ptcnet.Capacitor",
    "description": "This part extends ElectricalParts entity.",
    "parent": {
        "name": "ElectricalParts"
    },
    "attributes": [
```

```
{
     "name":"Capacitance",
     "internalName":"Capacitance",
     "type":"String"
}


    ]
}

Sample request to create WTPart with soft type 'Capacitor':
{
"@odata.type": "PTC.ProdMgmt.Capacitor",
"Name":"TestWTPart_002",
"Number":"TestWTPart_002",

"AssemblyMode": {
      "Value": "component",
      "Display": "Component"
     },
    "Context@odata.bind": "Containers('OR:wt.pdmlink.PDMLinkProduct:48788507')"
}
```

Extending Document Management Domain to Add a Soft Attribute

This example shows you how to extend the Document Management domain to add a soft attribute on a `WTDocument` soft type.

To extend the domain to add the soft attribute, create a custom configuration file `DocumentsExt.json` at `<Windchill>/codebase/rest/custom/domain/DocMgmt/v2/entity`.

```
{
    "extends": "Document",
    "description": "This config extends Documents.json.",
    "attributes": [

{
         "name":"ODATASTR1",
         "internalName":"ODATASTR1",
         "type":"String"
        },
        {
  "name":"ODATAINT1",
  "internalName":"ODATAINT1",
  "type":"Int16"
```

```
        },
        {
         "name":"ODATAFPN1",
         "internalName":"ODATAFPN1",
         "type":"Double"
        },
        {
         "name":"ODATABOOL1",
         "internalName":"ODATABOOL1",
         "type":"Boolean"
        },
        {
         "name":"ODATADATE1",
         "internalName":"ODATADATE1",
         "type":"DateTimeOffset"
        }
     ]

}
```

To create a `WTDocument` with these extended soft attributes use the following request:

```
POST /Windchill/servlet/odata/v2/DocMgmt/Documents HTTP/1.1
{
    "Name": "Test1",
    "Description": "Test1_Desc",
    "Title": "Test1_Title",

    "ODATASTR1": "This is String attribute",
    "ODATAINT1": 1,
    "ODATAFPN1": 1.555,
    "ODATABOOL1": true,
    "ODATADATE1": "2017-10-09T09:42:39Z",

"Context@odata.bind":
"Containers('OR:wt.pdmlink.PDMLinkProduct:48788507')"

}
```

A

Summary of Changes for Windchill REST Services 1.5

Changes in Windchill REST Services 1.5 are described in this section.

API Catalog for Windchill REST Services Endpoints

The Windchill REST Services release includes an API catalog, which is a developer document. The catalog is a webpage that is accessible from the Windchill user interface. The catalog is the Swagger specification of the endpoints available in Windchill REST Services. You can also interactively execute the HTTP operations on the endpoint URLs. The endpoints and operations are listed for every version of the domain.

The catalog can be extended. Any custom domains that you add can be enabled in the catalog.

Changes in Annotation

The changes in annotation follow:

- A new annotation `PTC.SecurityLabel` is available. It is applied to entity properties that are security labels in Windchill.
- The `PTC.UpdateableViaAction` annotation is applied to bindings of navigation properties that cannot be updated using the PATCH operation. You can update the navigation property using the `UpdateCommonProperties` action.

Creator and Modifier Navigation Properties

`Creator` and `Modifier` navigation properties are available for all the Windchill entities which support creator and modifier attributes. They contain all the information about the user who created or modified the entity.

Enhancements in the GetStructure Action

The enhancements in the `GetStructure` action are:

- Support for navigation criteria—The function supports navigation criteria as the filter criteria to return the CAD structure. You can specify the navigation criteria or the ID of the navigation criteria in the request payload.
- Support for Visualization URLs—The function returns `PVTreeId` and `PVParentTreeId` URLs which contain the occurrence paths to the components. These URLs can be used to work with the Visualization tree.

Enhancements in the PTC Change Management Domain

The enhancements in PTC Change Management domain follow:

- Navigation properties `ChangeAdministratorI`, `ChangeAdministratorII`, and `VarianceOwners` are supported. These navigation properties contain details about change administrator I, change administrator II, and variance owners.
- The `OrganizationID` attribute is not supported.

Enhancements in the PTC Product Management Domain

Clients can access assigned expressions and expression aliases. Assigned expressions are accessible from `Part`, `PartUse`, and `UsageOccurrence` entities. The expressions are assigned to these entities. Use the `GetAssignedExpression()` function and `GetAssignedExpressions` action to retrieve assigned expression from single and multiple objects respectively. Both dependent and independent expression modes are supported. Basic and advanced types of expressions are supported.

Modify the Organization for Parts and Documents Using the UpdateCommonProperties Action

The `UpdateCommonProperties` action is updated to enable you to modify the organization associated with parts and documents. The `Organization` navigation property is annotated with `PTC.UpdateableViaAction` to indicate that this property can be updated using only an action.

Navigation Property AllPrimaryContents Available for CADDocument Entity

A new navigation `AllPrimaryContents` is added in the `CADDocument` entity in the PTC CAD Document Management domain. Use the navigation to get the details of the primary content associated with the CAD Document.

The navigation `PrimaryContent` is not supported for `CADDocument` entity.

Navigation Properties Available in the BOP Entity

The following navigation properties are available in the BOP entity. The `GetBOP` action can retrieve all the information related to operations and their object in one call.

- `ConsumedOperatedOnParts`
- `ConsumedParts`
- `ConsumedProcessMaterials`
- `ConsumedSkills`
- `ConsumedStandardCCs`
- `ConsumedToolings`
- `ConsumedWorkCenters`
- `DescribedByDocuments`
- `ReferenceDocuments`
- `DownloadUrls`
- `Representations`

Navigation Property to Support Alias Expression

A new navigation, `OptionPoolAliases`, is added to the `ProductContainer` and `LibraryContainer` entities in the PTC Data Administration domain. Use the navigation to get the expression aliases that are part of the option pool in the product and library containers.

New Domains

The following PTC domains are added:

- PTC Regulatory Master domain provides access to regulatory submission capabilities of Windchill.
- PTC Workflow domain provides access to workflow capabilities of Windchill.

Retrieving the Bill of Process (BOP) Using Navigation Criteria

The `GetBOPWithInlineNavCriteria()` action returns the bill of process (BOP) for the process plan structure for the specified navigation criteria. The action is added in the PTC Manufacturing Process Management domain.

Retrieving the Consumed Objects of an Operation Using Navigation Criteria

The `GetConsumedWithInlineNavCriteria()` action returns the object associated to a consuming operation for the specified navigation criteria. The action is added in the PTC Manufacturing Process Management domain.

Retrieving the Windchill Metadata

The `GetWindchillMetaInfo` function is added in the PTC Common domain. The function is available in all the domains that import the PTC Common domain. The function retrieves the Windchill metadata for OData entity types and properties that are available in the domain from which the function is called. The function returns internal names and localized display names for Windchill types and properties.

securityLabeled Capability Added to the inherits Property

A new capability `securityLabeled` is available in the `inherits` property. You must inherit this capability to show the security labels of the entity. The security labels appear as properties of the entity. In the metadata, the properties representing a security label are annotated with `PTC.SecurityLabel` and `PTC.ReadOnly`.

Support for $count Query Parameter

`$count` is supported as a query parameter. When `$count` is specified in the URL as a query parameter, it returns the count of items in the collection being requested. Other uses of `$count` are not supported.

Support for Using Some Navigation Properties in the $filter and $orderby Expressions

You can specify some navigation properties while forming `$filter` and `$orderby` expressions.

Use the following navigation properties to filter or sort items in entity sets. Each property is followed by entity sets for which it is supported.

- `Context`—Change objects, parts, and documents
- `Organization`—Change objects, parts, and documents
- `Folder`—Parts and documents
- `Attachments`—Change objects and documents
- `PrimaryContent`—Documents
- `Representation`—Parts
- `AffectedObjects`—Change objects
- `AffectedLinks`—Change objects

The configuration property, `traversal`, is added in the `navigations` section of the <Entity JSON> file. This property specifies a traversal string for navigating between two entities.

Support for Configuration Specifications for CAD Documents

The PTC Navigation Criteria domain supports the following configuration specifications for CAD documents:

- `EPMDocBaselineConfigSpec`
- `EPMDocPromotionNoticeConfigSpec`
- `EPMDocStandardConfigSpec`
- `EPMDocAsStoredConfigSpec`

Support for subtypeable and softattributable Capabilities in the PTC Regulatory Master Domain

The PTC Regulatory Master domain supports the `subtypeable` and `softattributable` capabilities of Windchill for the `RegulatorySubmission` entity.

Support for Webhook Subscriptions

Subscription to events using the PTC Event Management domain is supported for the following domains:

- PTC Data Administration domain
- PTC Change Management domain
- PTC NC (Nonconformance) domain
- PTC CAPA domain
- PTC Customer Experience Management (CEM) domain

B

Summary of Changes for Windchill REST Services 1.4

Changes in Windchill REST Services 1.4 are described in this section.

Actions for Multiple Objects

The following actions take multiple objects as input parameters.

- `Create<EntitySet_name>`—Available for PTC Product Management and PTC Document Management domains.
- `Update<EntitySet_name>`—Available for PTC Product Management and PTC Document Management domains.
- `Delete<EntitySet_name>`—Available for entities in a domain where `"multiOperations": "DELETE"` is specified in the `Entity JSON` file.
- `Revise<EntitySet_name>`—Available for entities that inherit the `versioned` capability in a domain.

For example, `CreateParts()`, `UpdateParts()`, and so on are available in PTC Product Management domain. Similarly, `CreateDocuments()`, `UpdateDocuments()`, and so on are available in PTC Document Management domain.

classifiable Capability Available Out-of-the-Box

`classifiable` inheritance is available out-of-the-box. Explicit configuration by users is not be required.

Enhancements in the PTC Change Management Domain

Enhancements in change management are described below:

- The following navigation properties are available:
  - `ResultingObjects`
  - `ResultedByObjects`
  - `ResultingLinks`
  - `ResultingByLinks`
  - `UnincorporatedLinks`
  - `UnincorporatedByLinks`
- All the change management entities support the following attributes:
  - Change versions with their details.
  - Lifecycle template which is used to create the change object.
  - Change administrators with their details.

> **Note**
>
> Windchill REST Services currently supports only Change Admin I and Change Admin II.

  - Complexity of the change object.
  - Icon details.

Navigation Property to Support Assigned Option Set

The navigation property `AssignedOptionSet` retrieves the option set assigned to containers and parts. It is available in the following domains:

- PTC Data Administration domain—For product and library containers
- PTC Product Management domain—For parts

Navigation Property to Support Option Pool Items

The navigation property `OptionPool` retrieves the option pool items, option groups, and options that are associated with a product or library container. The navigation property is available for product and library containers in the PTC Data Administration domain.

New Domains

The following PTC domains are added:

- PTC Product Platform Management domain, which provides access to Options and Variants capabilities in Windchill.
- PTC CAD Document Management domain, which provides access to CAD data management capabilities of Windchill.
- PTC Event Management domain, which provides access to the webhook subscription capabilities of Windchill.
- PTC Supplier Management domain, which provides access to the supplier management capabilities of Windchill. Some supplier management objects, such as, supplier part, manufacturer part and so on are available in the PTC Product Management domain.

Retrieving Bill of Materials Along with Path Details for Occurrences

The action `GetPartStructure` returns the bill of materials (BOM) for a product structure along with path details for occurrences. The action supports **Option Filter**.

The action is available in the PTC Product Management domain.

Support for Filtering for Parts and Documents Based in Container or Organization Name

`$filter` supports filtering a collection of parts and documents based on the name of:

- Organization
- Container

Support for OData Preferences return=representation and return=minimal

Windchill REST Services supports OData preferences `return=representation` and `return=minimal` in headers.

When you set the preference `return=representation`, the request returns the modified content in the body of the response along with the HTTP status code.

The preference `return=minimal` does not return the modified content in the body of the response. It only returns the appropriate HTTP status code.

Support for Option Filter

The following domain and actions support **Option Filter**:

- PTC Navigation Criteria domain
- `GetBOM` action
- `GetBOMWithInlineNavigationCriteria` action

The `NavigationCriteria` entity retrieves option filter, which is associated with the navigation criteria. The `OptionFilter` complex type is used to retrieve information about option filter.

Support for subtypeable and softattributable Capabilities in the PTC CAD Document Management Domain

The PTC CAD Document Management domain supports the `subtypeable` and `softattributable` capabilities of Windchill for the following entities:

- `subtypeable`—`CADDocument` entity
- `softattributable`—`CADDocument`, `CADDocumentUse` and `CADDocumentReference` entities

C

Summary of Changes for Windchill REST Services 1.3

Changes in Windchill REST Services 1.3 are described in this section.

Actions for Multiple Objects

The following actions take multiple objects as an input parameter: These actions are available for all the entities that inherit the `workable` capability.

- `CheckIn<EntitySet_name>`
- `CheckOut<EntitySet_name>`
- `UndoCheckOut<EntitySet_name>`
- `Delete<EntitySet_name>`

For example, `CheckInParts()`, `CheckOutParts()`, and so on are available in PTC Product Management domain. Similarly, `CheckInDocuments()`, `CheckOutDocuments()`, and so on are available in PTC Document Management domain.

Batch Requests Support References to Entity and Property Value

Batch requests support references to entity and property values between requests and responses of different parts of a batch. Change set request can reference an entity and property value from a change set of a previous batch request.

classifiable Capability Added to the inherits Property

A new capability `classifiable` is available in the `inherits` property. You must inherit the capability in the domain to classify the part entities.

Editing Common Attributes of an Entity

The property *hasCommonProperties* in the `<Entity JSON>` is used to specify if an entity contains Windchill attributes that are common to objects.

The parameter *common* in the `<Entity JSON>` specifies if the parameter is a Windchill attribute that is common for objects and can be updated.

The action `UpdateCommonProperties` available in the PTC Product Management and PTC Document Management domain is used to update the common attributes.

Enhancements in Classification

Enhancements in classification are described below:

- PTC Classification Structure (ClfStructure) domain
  - `Required`—The property `Required` is added to the `ClfAttributeType` complex type. This property specifies if the attribute is a mandatory attribute required for classification.
  - `GetClassificationNodeInfo()`—The function retrieves the information about classification attributes for the specified classification node.
- PTC Common (PTC) domain—Two new complex types `ClassificationInfo` and `ClassificationAttribute` are available. These complex types are used to represent classification binding and classification attributes.

Enhancements in the PTC Change Management Domain

Enhancements in change management are described below:

- The PTC Change Management domain supports the `subtypeable` and `softattributable` capabilities of Windchill for all entities.
- The following navigation properties are available:
  - `AffectedLinks`
  - `AffectedObjects`
  - `AffectedByLinks`
  - `AffectedByObjects`
  - `Attachments`
- The following navigation properties now support multiple objects:
  - `ProcessLinks`
  - `ProcessObjects`
  - `ReferenceLinks`
  - `ReferenceObjects`

Entity and Functions Moved from PTC Common Domain to PTC Visualization Domain

The following entity and functions have been moved from PTC Common domain to PTC Visualization domain:

- `Representation` entity
- `GetPVZ()` function
- `GetDynamicStructureRepresentation()` function

LicenseGroup Entity Added in PTC Principal Management Domain

The entity `LicenseGroup` is added in the PTC Principal Management domain. The entity represents license groups of Windchill.

New Domains

The following PTC domains are added:

- PTC Navigation Criteria domain, which provides access to filters available in a part structure in Windchill.
- PTC Parts List Management domain, which provides access to parts list and parts list item in Windchill.
- PTC Service Information Management domain, which provides access to objects and structures of Windchill Service Information Manager.
- PTC Visualization domain, which provides access to visualization services of Windchill.
- PTC Audit domain, which provides access to auditing capabilities available in the Quality product of Windchill.
- PTC Customer Experience Management domain, which provides access to the Windchill customer experience capabilities.

Retrieving Bill of Materials Using Navigation Criteria

The action `GetBOMWithInlineNavCriteria()` returns the bill of materials (BOM) for the product structure for the specified navigation criteria. The action is added in the PTC Product Management domain.

Retrieving Components List for a Part Structure

The action `GetPartsList` returns a consolidated list of components, their quantities and values, for the specified part structure. The action is added in the PTC Product Management domain.

Support Create, Update, and Delete Operations for folder Entity

Create, update, and delete operations are supported for `folder` entity, which is available in the PTC Data Administration domain.

Support for $orderby Query Parameter

`$orderby` query parameter is supported for some primitive types and complex types. The `SortRestrictions` annotation is used to specify the properties that must not be used in the `$orderby` expressions.

Support for subtypeable and softattributable Attributes in the PTC Manufacturing Process Management Domain

The PTC Manufacturing Process Management domain supports the `subtypeable` and `softattributable` capabilities of Windchill.

D

Summary of Changes for Windchill REST Services 1.2

Changes in Windchill REST Services 1.2 are described in this section.

Creating Parts and Documents in a Different Organization

If the preference **Expose Organization** in the **Preference Management** utility is set to `Yes` in Windchill, you can specify a different organization, when creating parts and documents.

Get Information About All Life Cycle States in Windchill

The function `GetAllStates()` returns a list of life cycle states which are available and can be selected in Windchill. The function is available in the PTC Common domain.

Get Information About Life Cycle States for Objects in Windchill

The function `GetValidStateTransitions()` returns the life cycle states that the entity can transition from its current state. The function is available for all the entities which are life cycle managed.

Navigation Properties to Support Service Information Manager Translation

Navigation properties to support Service Information Manager Translation are available in the PTC Dynamic Document Management domain.

New Domains

The following new PTC domains have been added:

- PTC Manufacturing Process Management domain, which provides access to the manufacturing process management capabilities (MPM) of Windchill.
- PTC Change Management domain, which provides access to the change management capabilities of Windchill.
- PTC Classification Structure domain, which provides access to the classification structure and classification nodes in Windchill.
- PTC Saved Search domain, which provides access to saved searches in Windchill.

Multivalued Attributes Supported for Structural Properties

The parameter *isCollection* supports multivalued attributes, such as, complex types, for structural properties.

nonFilterable Parameter Added for Entity Attributes

The parameter *nonFilterable* is available as an annotation for attributes. It specifies if you can use a `$filter` expression to query an attribute.

Retrieve Dynamic Structure of a Creo View Representation

The function `GetDynamicStructureRepresentation()` returns a URL which you can use to download the dynamic structure of a Creo View representation. The function is available for all the parts and documents which have representations associated with them. The function is available in the PTC Common domain.

Retrieve PVZ File from a Creo View Representation

The function `GetPVZ()` returns a ZIP file, which contains OL, PVS, PVT, and other Creo View files. The function is available in the PTC Common domain.

Set Life Cycle State for an Entity

The action `SetState()` sets a valid life cycle state for an entity. The action is available for all the entities which are life cycle managed.

Enhancements to the inherits Property

Capabilities added to the `inherits` property are described below:

- `subtypeable`—Subtypes that are available in Windchill are enabled in the domains of Windchill REST Services.
- `softattributable`—Attributes of entity types and subtypes that are available in Windchill are enabled in the domains of Windchill REST Services.

Update Comments and Description Attributes for an ContentItem Entity

You can use PATCH requests to update the `Comments` and `Description` properties for the `ContentItem` entity.

URLs Available in the Representation Entity

The `Representation` entity returns the following URLs in the body of the response:

- `CreoViewURL`—Contains the URL that starts the Creo View application.
- `3DThumbnailURL`—Contains the URL for 3D thumbnail.
- `2DThumbnailURL`—Contains the URL for 2D thumbnail.
- `AdditionalFiles`—Contains the URL for additional files, which are non-native Creo View files. These files are associated with the specified representation.

WindchillEntity Added in the PTC Common Domain

A new entity `WindchillEntity` has been added in the PTC Common domain. The entity represents Windchill objects types that are not available in Windchill REST Services.

E

Summary of Changes for Windchill REST Services 1.1

Changes in Windchill REST Services 1.1 are described in this section.

Getting Information About Windchill Constraints

You can get information about the constraints applied to an entity property. The new function `GetConstraints()` returns information related to constraints for an attribute. The function has been added in the PTC Data Administration domain.

Getting the Value of a Nonce Token

You can get the value of a Nonce token. The new function `GetCSRFToken()` returns the nonce value. The function has been added in the PTC Common domain.

Entity Information for PTC Document Management Domain and PTC Common Domain

A new entity `ContentInfo` has been added to the PTC Document Management domain. It contains information about content that will be used in Stage 3 document upload.

The following entities have been moved from PTC Document Management domain to PTC Common domain:

- `ContentItem`
- `ExternalStoredData`
- `URLData`
- `ApplicationData`

New Domains

The following new PTC domains have been added:

- PTC Dynamic Document Management Domain, which provides access to the dynamic document capabilities of Windchill.
- PTC Quality Domains, which provides access to the Quality Management Services of Windchill.
- PTC Info*Engine System Domain, which provides access to the Info*Engine tasks of Windchill.
- PTC Factory Domain, which provides access to manufacturing data management capabilities of Windchill.

Support for $filter expression for Navigation and Expansion

The `$filter` expressions are supported for filtering navigation and expanded navigation properties.

Support for Using DateTimeOffset Attribute in Filter Expressions

Windchill REST Services supports the OData attribute type `DateTimeOffset`. This release enables you to query with date in `$filter` expressions.

Support for OData Prefer Header

Windchill REST Services supports `odata.maxpagesize` preference in headers. The preference sets the value for the number of items to be returned in a collection for a response.

Support for Separate Configuration Paths for Domain Authors

Domain authors can now maintain their configurations in folder paths separate from PTC configurations.

Support for FilterRestrictions Annotation on Properties that Impact Performance

The framework now automatically annotates properties that impact performance. For this, the `FilterRestrictions` annotation from Capabilities Vocabulary in OData is used.

Entity properties that are nonpersistant or are calculated by Windchill are automatically annotated by the framework.

F

Version Changes in Domains

Some important version changes in domains are described in this section.

Windchill REST Services 1.5

The following version changes are implemented in Windchill REST Services 1.5:

Domain	Version Number	Reason for Versioning
PTC Manufacturing Process Management domain (MfgProcMgmt)	v2	To import the latest version of the PTC Navigation Criteria domain (NavCriteria). This domain enables you to specify the navigation criteria as the input parameter to the actions available in the PTC Manufacturing Process Management domain.
PTC CAD Document Management Domain (Change)	v2	• To support subscription to events using the PTC Event Management domain. • `OrganizationID` attribute is not supported.

Windchill REST Services 1.4

The following version changes are implemented in Windchill REST Services 1.4:

Domain	Version Number	Reason for Versioning
PTC Saved Search domain (SavedSearch)	v2	To import the latest version of all the domains that are required in PTC Saved Search domain.

Windchill REST Services 1.3

The following version changes are implemented in Windchill REST Services 1.3:

Domain	Version Number	Reason for Versioning
PTC Common domain (PTC)	v2	• The entity `Representation` is moved to the PTC Visualization domain. • The functions `GetDynamicStructureRepresentation()` and `GetPVZ()` are available with the `Representation` entity in the PTC Visualization domain. However, the navigation property `Content` is no longer supported.
PTC Product Management domain (ProdMgmt)	v3	• Use the `Representation` entity available in the PTC Visualization domain. • The support for navigation property `Content` is removed from `Representation` entity. Clients cannot navigate from the `Representation` entity to its content. • The entity `NavigationCriteria` is moved to PTC Navigation Criteria domain. • Action and parameter names are changed to follow the camel case

Domain	Version Number	Reason for Versioning
		format.
PTC Document Management domain (DocMgmt)	v3	• As a result of version change of the PTC Common domain (PTC). • It is mandatory to specify the `Name` attribute for the `Document` entity. • Action and parameter names are changed to follow the camel case format.
PTC Principal Management domain (PrincipalMgmt)	v3	As a result of version change of the PTC Common domain (PTC).
PTC Data Administration domain (DataAdmin)	v3	• As a result of version change of the PTC Common domain (PTC). • Action and parameter names are changed to follow the camel case format.

G

HTTP Status Codes Returned by Windchill REST Services Responses

Windchill REST Services returns appropriate HTTP status code along with its responses. If an error occurs, Windchill REST Services returns an HTTP status code along with an appropriate error message.

The table shows you the HTTP status codes that are returned along with Windchill REST Services responses.

HTTP Status Code	HTTP Status Code Definition	Windchill REST Services Description
200	HTTP request is successful	Windchill REST Services returns this code when an end point successfully returns entities or entity collections.
201	HTTP request successfully created a resource	Windchill REST Services returns this code when a POST request to a resource or an action successfully creates an entity. The response returns the newly created entity.
204	HTTP request is successful, but the service does not return any	Windchill REST Services returns this code when an entity instance is

HTTP Status Code	HTTP Status Code Definition	Windchill REST Services Description
	response.	successfully updated or deleted with: • PUT, PATCH, and DELETE requests • POST request to action Windchill REST Services returns 204 HTTP code with no response.
400	Bad HTTP request	Windchill REST Services returns this code when the client request is malformed or cannot be processed by the server.
403	HTTP request is formed correctly, but is denied by the service	Windchill REST Services returns this code when the user of the request does not have necessary permissions to process the request.
404	Requested resource is not found	Windchill REST Services returns this code when the domain versions, navigations, or entity instances being requested are not available.

HTTP Status Code	HTTP Status Code Definition	Windchill REST Services Description
500	Internal server error	Windchill REST Services returns this code when the processing of the request fails due to unexpected runtime conditions.
501	Functionality has not been implemented on the server	Windchill REST Services returns this code when URLs are valid OData URLs, but the functionality has not been implemented. For example, in Windchill REST Services some operations in `$filter` parameter are not implemented. If the client requests for such operations in the URL, Windchill REST Services returns the 501 HTTP code.

H

Summary Javadoc for NavigationProcessorData

```
public class NavigationProcessorData
extends EntityProcessorData
```

An `NavigationProcessor` parameter object used for navigation entity associations.

Supported API: `true`

Extendable: `false`

Constructor Summary

Constructors

Constructor and Description

```
NavigationProcessorData(AbstractEdmProvider provider,
EntityProcessorData sourceEntityData,
org.apache.olingo.server.api.uri.UriResourceNavigation uriResourceNavigation)
```

Initializes a new `NavigationProcessorData` instance for the given requested entity and navigation.

Supported API: `true`

All MethodsInstance MethodsConcrete Methods

Modifier and Type	Method and Description
`EntityConfig`	`getSourceEntityConfig()` Returns the source entity configuration which contains information such as name, description, collection name, attributes, operations, and so on, about the entity. Supported API: `true`

`java.util.Collection`	`getSourceObjects()`
	Returns the source objects for the navigation.
	For example, in the following request to navigate `Uses` for a part, the source object is the part with `id` 396690:
	`odata/ProdMgmt/Parts('OR:wt.part.WTPart:396690')/Uses`
	Supported API: `true`

`org.apache.olingo.commons.api.edm.EdmEntityType`	`getTargetEntityFilteredType()` Returns the specific target entity which is of filtered type. You must navigate to this entity. Supported API: `true`
`org.apache.olingo.commons.api.edm.EdmEntityType`	`getTargetEntityType()` Returns the target entity to which you must navigate. Supported API: `true`

All MethodsInstance MethodsConcrete Methods (continued)

Modifier and Type	Method and Description
`void`	`setSourceEntityConfig(EntityConfig sourceEntityConfig)` Sets the configuration of the source entity. Supported API: `true`
`void`	`setSourceObjects(java.util.Collection objects)` Sets the source object for the navigation. Supported API: `true`

Summary Javadoc for EntityProcessorData

```
public class EntityProcessorData
extends java.lang.Object
implements java.lang.Cloneable
```

An `EntityProcessor` parameter object used for creating, reading, updating and deleting entities.

Supported API: `true`

Extendable: `false`

Constructor Summary

Constructors

Constructor and Description

`EntityProcessorData(EntityConfig config)`

Supported API: `true`

All MethodsStatic MethodsInstance MethodsConcrete Methods

Modifier and Type	Method and Description
`org.apache.olingo.commons.api.edm.EdmEntitySet`	`getEdmEntitySet()` Returns the entity set, that is, the collection of entity type instances, which are generated from the request. Supported API: `true`
`org.apache.olingo.common`	`getEdmEntitySet(EntityType`

All MethodsStatic MethodsInstance MethodsConcrete Methods (continued)

Modifier and Type	Method and Description
`s.api.edm.EdmEntitySet`	`entityType)` Returns the entity set for the given entity type, such as, Part or Document. Supported API: `true`
`org.apache.olingo.common s.api.edm.EdmEntityType`	`getEdmEntityType()` Returns the entity type data generated from the request. Supported API: `true`
`EntityConfig`	`getEntityConfig()` Returns the requested entity configuration which contains information such as, name, description, collection name, attributes, operations, and so on, about an entity. Supported API: `true`
`java.lang.String`	`getEntitySetName()` Returns the requested entity set name, such as, Parts or Documents. Supported API: `true`
`EntityType`	`getEntityType()` Returns the requested entity type, such as, Part or Document. Supported API: `true`
`javax.servlet.http.Htt pServletRequest`	`getHttpRequest()` Returns the original `HttpServletRequest`. Supported API: `true`
`java.util.Locale`	`getLocale()` Returns the locale for the request. Supported API: `true`
`org.apache.olingo.server.a pi.OData`	`getOdata()` Returns the OData for the request. Supported API: `true`

All MethodsStatic MethodsInstance MethodsConcrete Methods (continued)

Modifier and Type	Method and Description
`org.apache.olingo.server.api.ODataRequest`	`getRequest()` Returns the OData request. Supported API: `true`
`java.lang.String`	`getSelectedURLContext()` Returns the requested select URL context for the specified entity type. Supported API: `true`
`org.apache.olingo.server.api.ServiceMetadata`	`getServiceMetadata()` Returns the `ServiceMetadata` for the request. Supported API: `true`
`org.apache.olingo.server.api.uri.UriInfo`	`getUriInfo()` Returns the URI information from the request. Supported API: `true`
`void`	`setEdmEntitySet(org.apache.olingo.commons.api.edm.EdmEntitySet edmEntitySet)` Sets the entity set data, which is generated from the request. Supported API: `true`
`void`	`setEdmEntityType(org.apache.olingo.commons.api.edm.EdmEntityType type)` Sets the entity type data, which is generated from the request. Supported API: `true`
`void`	`setLocale(java.util.Locale locale)` Sets the locale for the request. Supported API: `true`
`void`	`setOdata(org.apache.olingo.server.api.OData odata)`

All MethodsStatic MethodsInstance MethodsConcrete Methods (continued)

Modifier and Type	Method and Description
	Sets the OData for the request. Supported API: `true`
`void`	`setServiceMetadata(org.apache.olingo.server.api.ServiceMetadata serviceMetadata)` Sets the `ServiceMetadata` for the request. Supported API: `true`
`void`	`setUriInfo(org.apache.olingo.server.api.uri.UriInfo uriInfo)` Sets the URI information for the request. Supported API: `true`

Index

D

E

F

H

I

J

N

O

P

Q

R

S

T

U

V

W