TEAMCENTER

Structure Management — Deployment and Administration

Teamcenter 2412

SIEMENS

About Siemens Digital Industries Software

Siemens Digital Industries Software is a global leader in the growing field of product lifecycle management (PLM), manufacturing operations management (MOM), and electronic design automation (EDA) software, hardware, and services. Siemens works with more than 100,000 customers, leading the digitalization of their planning and manufacturing processes. At Siemens Digital Industries Software, we blur the boundaries between industry domains by integrating the virtual and physical, hardware and software, design and manufacturing worlds. With the rapid pace of innovation, digitalization is no longer tomorrow’s idea. We take what the future promises tomorrow and make it real for our customers today. Where today meets tomorrow. Our culture encourages creativity, welcomes fresh thinking and focuses on growth, so our people, our business, and our customers can achieve their full potential.

Support Center: **support.sw.siemens.com**

Send Feedback on Documentation: **support.sw.siemens.com/doc_feedback_form**

Contents

Configure product structure creation and updates

Configure BOM line properties

Set up import and export of structures

Configure structure effectivity

Administer solution variants

Configure Product Configurator variants

Configure classic variants

Configure modular variants

Configure incremental changes for the rich client

1. Plan the structure management deployment and administration

The Teamcenter BOM Management suite of products helps users to manage product structures.

Before you begin

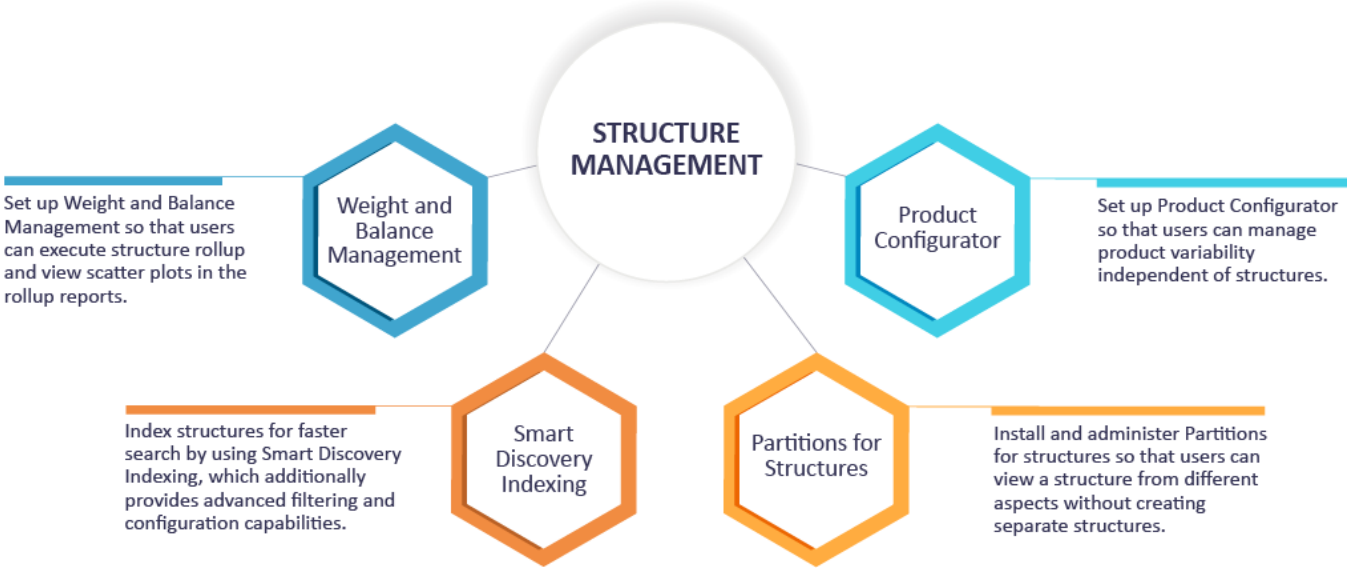
Ensure that your Teamcenter setup has the following applications:

Application	Why it is needed
Business Modeler IDE (BMIDE)	To configure Teamcenter for structure management.
Dispatcher	To run various user tasks on the background.
(Optional) Change Manager	To perform structure management tasks in the context of a change.
Lifecycle Visualization	To visualize products and to perform digital mock-ups on rich client.
Visualization Server (on Windows or Linux)	To visualize products and to perform digital mock-ups on Active Workspace.

Apart from these Teamcenter applications, you may need a CAD application, such as NX or CATIA, to manage the designs, and may need to integrate the application with Teamcenter.

Choose the required BOM products

First set up Teamcenter for structure management. Next, based on your business requirements, you can include other applications in your Teamcenter set up for additional structure management features. Each of these applications need different licenses. Ensure that the correct licenses are obtained and installed. Next, deploy and administer these products so that the business users can perform structure-related tasks in Teamcenter.



Where do I go from here?

Business User	
Manage structures on Active Workspace	See Structure Management on Active Workspace.
Manage product structures on the rich client	See Structure Management on the rich client.
Organize BOM content within partitions	See Partition Management on Active Workspace.
Administrator	
Set up Teamcenter for structure management	**Install** Teamcenter applications required for structure management and perform the tasks to **administer** structures so that users can manage structures.
Index structures for faster search	Index structures by using one of the following methods: • Smart Discovery Indexing • **Active Content Structure Indexing** Warning: The Active Content Structure indexing feature is soon going to be deprecated. You must index structure using this indexing only for massive model visualization. Else, it is recommended

	that you now start indexing structures by using Smart Discovery Indexing. • Cacheless Search
Configure structure search	Explore the search options and choose which ones to configure for your users.
Index structures for advanced filtering and configuration capabilities	Set up and administer Smart Discovery for Structures.
Deploy and administer Partitions for structures	Install and administer Partitions for Structures so that users can view a structure from different aspects without creating separate structures.
Deploy and administer Product Configurator	Set up Product Configurator so that users can manage the product variability independent of structures.
Control the behavior and appearances of Teamcenter for structures	Use preferences to control the various aspects of Teamcenter to suit the requirements at your site. For information about retrieving a list of preferences, see *Where can I get a list of preferences?* in *Teamcenter Preferences*.

2. Install Teamcenter applications for structure management

- Rich client

  Structure Manager does not require any other feature to be installed to make it available in Teamcenter rich client.

- Enable variability by using Product Configurator

  Select the following applications in the **Applications** task in Deployment Center or select the following features in TEM:

  Product Configurator Support for Structure Manager: Enables the use of Product Configurator to configure the variability of a product structure. This must be installed on both the Enterprise tier and the Client tier.

- Improve the performance of BOM expansion during revision configuration using revision rules

  You must install the **Teamcenter Revision Config Accelerator Service** service and make sure that it is running. To install this service:

  - In TEM, go to **Server Enhancements** > **Database Daemons** > **Teamcenter Revision Configuration Accelerator Service**.

    OR

  - In Deployment Center, go to the **Components** tab and add the **Database Daemon** component. Then, under **Database Daemon Services**, select the **Teamcenter Revision Configuration Accelerator Service** check box.

- Active Workspace

  Select the following applications in the **Applications** task in Deployment Center or select the following features in TEM:

  - **Active Content Structure**: Provides functionality and data model extensions necessary for indexing structure data.

  - **Active Content**: Adds structure search functionality to the client interface.

    Required if you select **Active Content Structure** when building the client web application.

- **Product Configurator Support for Active Content Structure**: Installs capabilities related to solution variants.

- **Solution Variant Support for Active Workspace**: Adds Product Configurator capabilities for Active Content Structure, such as creating and managing solution variant structures for configurable generic assemblies.

For more information about installing Teamcenter, see *Teamcenter Installation on Windows Using TEM* or *Teamcenter Installation on Linux Using TEM*.

For information about using Deployment Center, see *Deployment Center — Usage*.

> Note:
>
> You must install all server and client-side components to ensure proper functioning. For example, the **Product Configurator Support for Structure Manager** component must be installed on both the Enterprise tier and the Client tier so that Product Configuratorcan be used to configure the variability of a product structure.

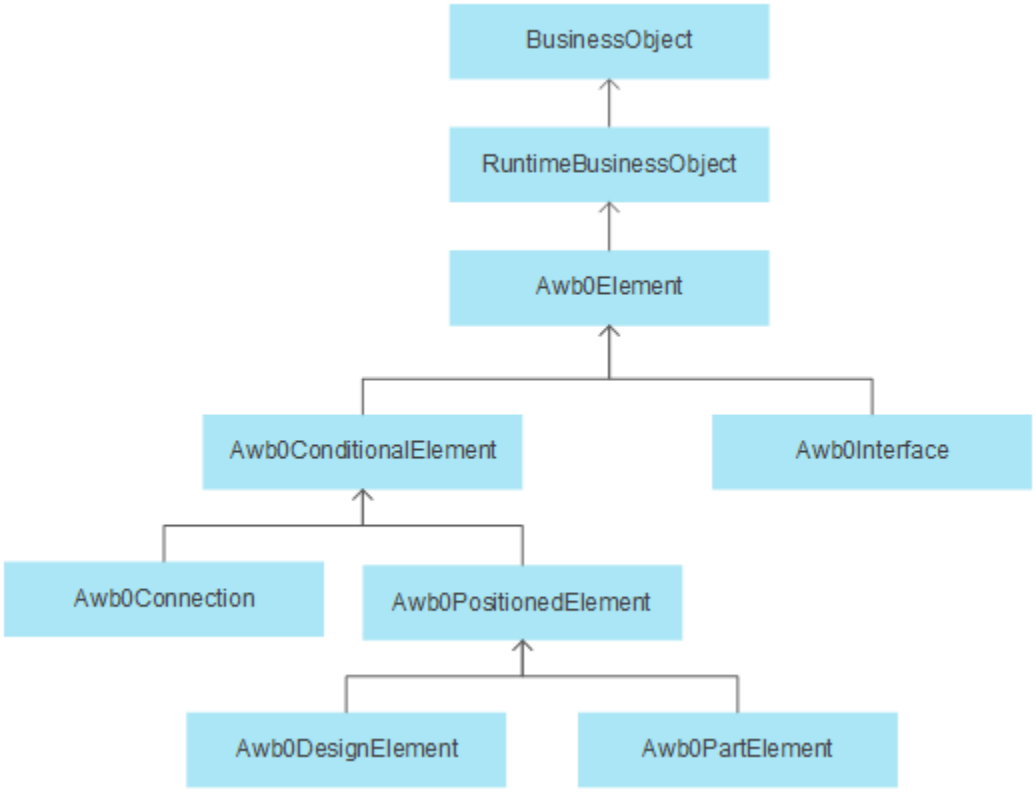
3. Extend the default Teamcenter data model for structures

Business objects required to extend Teamcenter for structures on Active Workspace

Active content provides a number of business objects to represent the occurrences returned by the various structure applications. **Awb0Element** is the root type and is abstract. The following table lists the default active content types that represent occurrence types for BVR structures.

Business object	Purpose
Awb0ConditionalElement	Represents an occurrence with effectivity and variant conditions. Generic design element (GDE) lines do not have effectivity or variant conditions; they should not be mapped to this type or its subtypes.
Awb0Connection	Represents an occurrence that does not have any associated geometry information.
Awb0DesignElement	Represents an occurrence with geometry. It provides the properties for managing geometry, for example, bounding box and transform. Objects of this type can be visualized in the viewers.
Awb0Element	Represents the root business object for all occurrence management objects. An element is identified by a name and can belong in a structure with a parent-child relationship. This is an abstract object and should not be mapped with any item revision type.
Awb0Interface	Represents the generic design element.
Awb0PartElement	Represents a part type and is mapped with the part revision. All the part-specific properties and behavior are associated to the part element.
Awb0PositionedElement	Represents an item revision type with geometry.

Hierarchy of business objects in the active content data model is shown in the figure:



Make BOMLine properties available on the Awb0Element for consistent data representation and accessibility

Making **BOMLine** properties available to **Awb0Element** ensures consistent data representation and accessibility. It also makes **BOMLine** data available in the context of **Awb0Element** and displays it correctly. You can map both out of the box and custom properties. However, runtime configuration properties are automatically mapped.

Prerequisites

- Check if the property is already mapped on **Awb0ConditionalElement** or one of its child types. For example, if a property is already mapped on **Awb0DesignElement**, do not map the same property on **Awb0ConditionalElement** to avoid an inconsistent data model state.

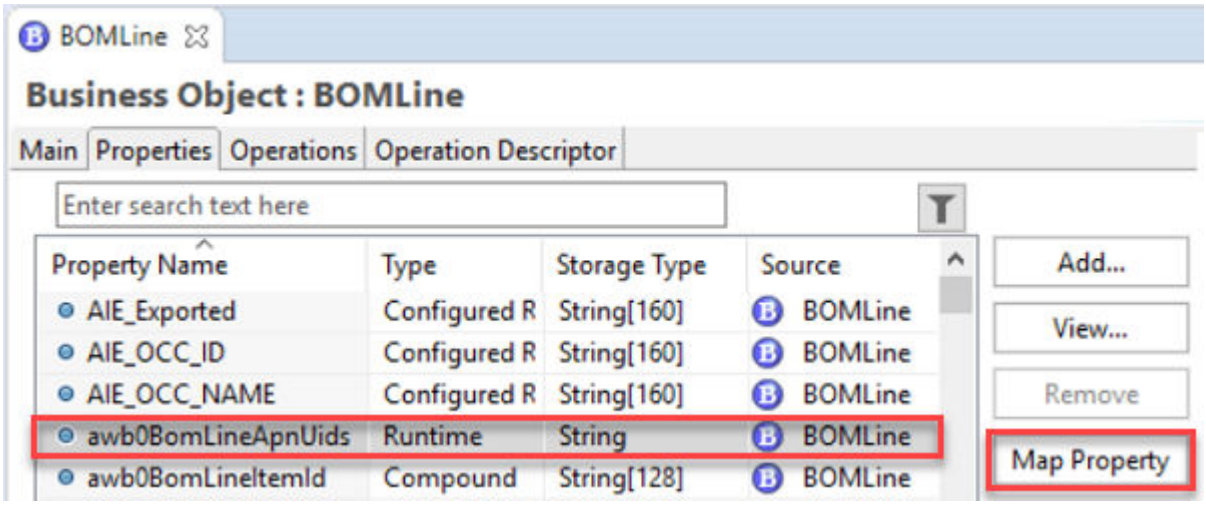
- You can create additional model elements as needed and attach them to the newly created property after mapping. For example, **LOV attaches**, **Property Formatter**, **Property Renderer**, and **Localization**.

Procedure

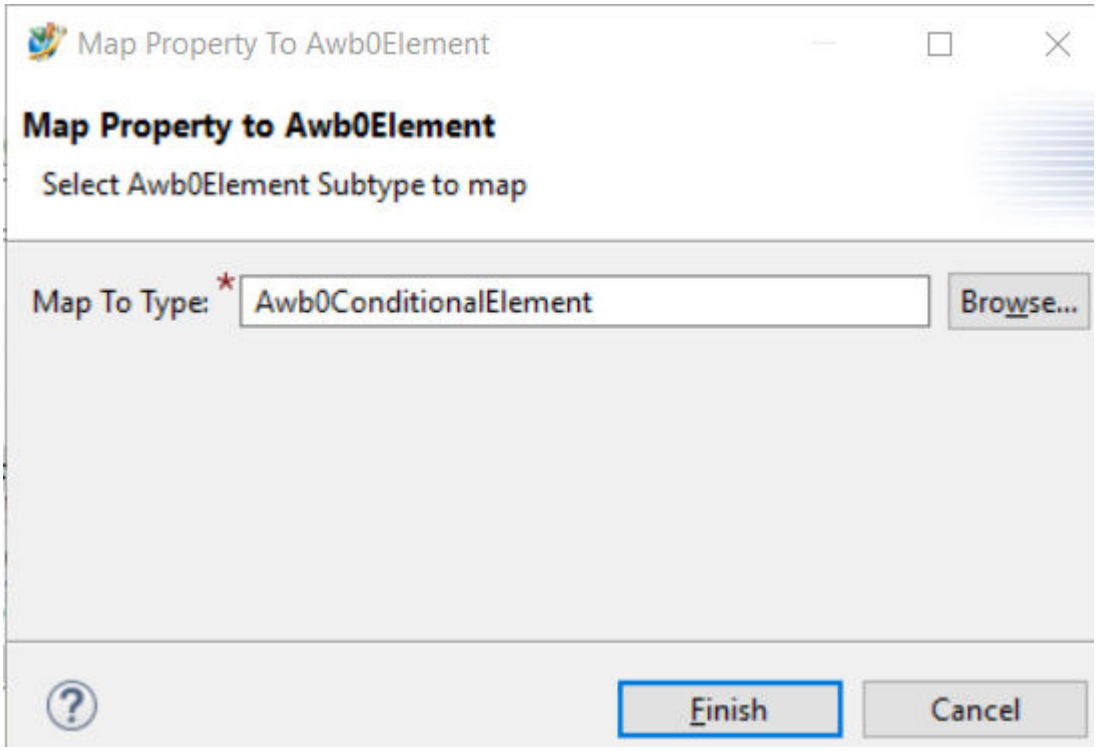
1. In the custom BMIDE template project, search and open the **BOMLine** business object.

2. In the **Properties** tab, search and select the property to map. For example, **awb0BomLineApnUids**.

3. Click the **Map Property** button.



4. In the **Map To Type** box, browse and select the **Awb0ConditionalElement** type.



5. Click **Finish**.

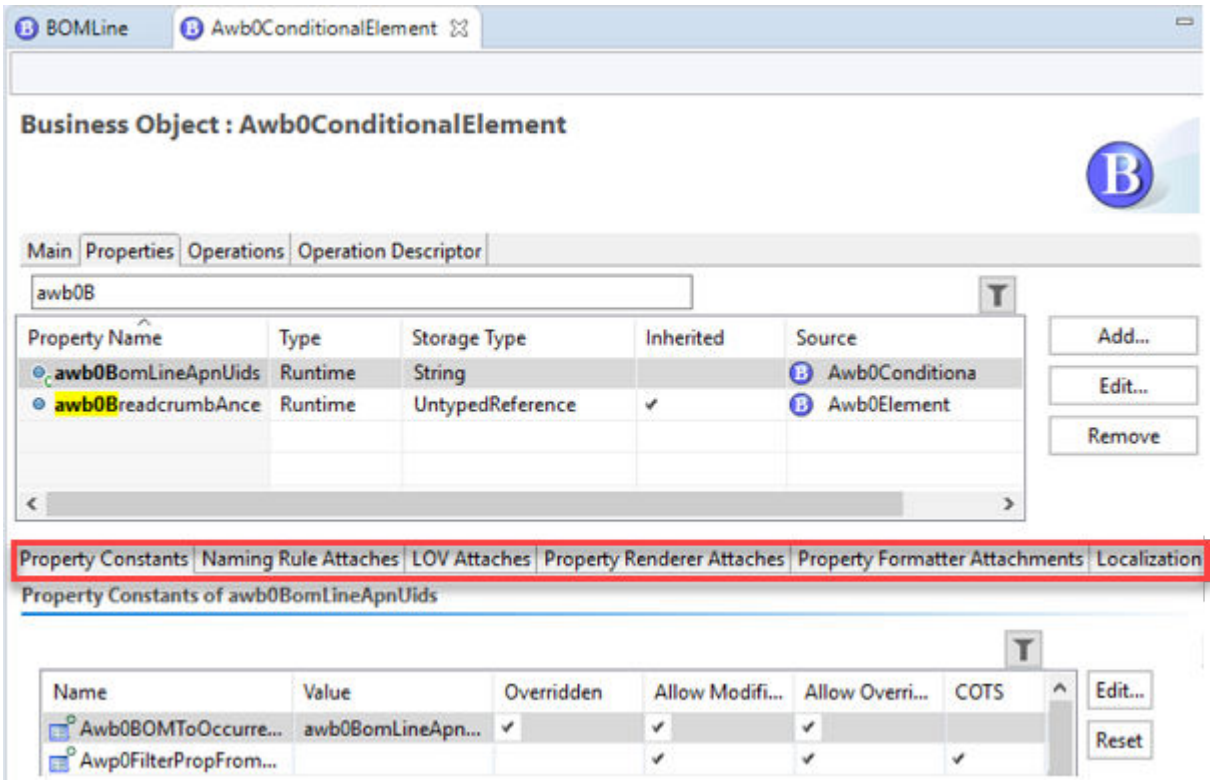
Now, the property is also available on the **Awb0ConditionalElement** with the same name and type as that of the **BOMLine** property.

The **Awb0BOMToOccurrence** property constant value of the property **awb0BomLineApnUids** on the **Awb0ConditionalElement** is set to the name of the existing BOMLine property.

The **LOV attaches**, **Property Renderer**, **Property Formatter**, and **Localization** data copied from the existing BOMLine property to the new property.

Or

You can add the custom model data that you created before mapping.



If you want to remove the newly created mapping, search and open the **Awb0ConditionalElement** business object, select the property (**awb0BomLineApnUids**) and click on the **Remove** button.

6. Save and deploy the template.

Mapping properties to occurrence properties

Domain-specific occurrences contain properties relevant to the specific domain. End users understand and interact with these domain-specific properties. These properties are mapped from the **BOMLine** or **ModelElement** type onto the occurrence. This mapping is provided by property constants defined in the Business Modeler IDE and the property constants are scoped to the **Awb0Element** type.

Default properties are provided on **Awb0Element** and its subtypes, but you can add custom properties necessary for your implementation.

All custom properties must be mapped to a property defined on the type specified in the **Awb0BOMToOccurrence** type constant. The property mapping is then achieved through the **Awb0BOMToOccurrence** property constant. The value of this property constant is inherited and can be overridden at any level.

For example, the **awb0BoundingBox** property on the **Awb0PositionedElement** business object has the value of **bl_bounding_boxes**. It also has the value **BOMLine** for the **Awb0BOMToOccurrence** type constant. Consequently, whenever the **awb0BoundingBox** property is requested on an **Awb0PositionedElement** object, the value is fetched from the **bl_bounding_boxes** property of the BOM line.

The mapping of some common default properties are listed below:

Business object	Property	Awb0BOMToOccurrence value
Awb0Element	**awb0Parent**	**bl_parent**
Awb0Element	**awb0Name**	**bl_line_name**
Awb0Element	**awb0ElementId**	
Awb0Element	**awb0Archetype**	**bl_revision**
Awb0ConditionalElement	**awb0ArchetypeEffFormula**	**bl_revision_effectivity**
Awb0ConditionalElement	**awb0ElementId**	**awb0BomLineItemId**
Awb0PositionedElement	**awb0BoundingBox**	**bl_bounding_boxes**
Awb0PositionedElement	**awb0Transform**	**bl_plmxml_abs_xform**

Default getter and setter methods are registered for all properties of **Awb0Element** and child business objects that have a mapping defined in the **Awb0BOMToOccurrence** property constant. You must provide a custom getter and setter for all properties that are not already mapped. For example, the **awb0NumberOfChildren** property specifies the number of child elements and does not have a value for the property constant; custom getter and setter methods are registered for it. You can use the same mechanism to register getter and setter methods for properties on custom **Awb0Element** subtypes.

Many configured runtime properties on **BOMLine** are derived from **Item**, **ItemRevision**, or another object. These are always of type **string**, irrespective of the type from which they are derived. Mapping to such a property will result in losing the type information about the property that the system requires for proper filtering of Solr results. To avoid this, instead of mapping to a configured runtime property, define a new compound property on the backing object and use the relations or reference properties to get the source property. This maintains the type information for the property so that it can be used in mapping.

For example, you may want to get the last modified date stored in the configured runtime property **bl_rev_last_mod_date** on the BOM line. Instead of using the bl_rev_last_mod_date property, consider

the **awb0RevisionLastModifiedDate** compound property on the BOM line. This uses the **bl_revision** property to access the item revision and you can get the **last_mod_date** property from there. An **awb0ArchetypeRevLastModDate** property is defined on the **Awb0DesignElement** business object and then mapped to the **awb0RevisionLastModifiedDate** property.

Applying custom BOMLine SOAs to the elements on Active Workspace

Active Workspace provides several business objects to represent the structure occurrences. In Active Workspace, **Awb0Element** is the root type. Each **Awb0Element** has a backing BOMLine. However, the BOMLine object is not directly accessible to the customizers of Active Workspace. Therefore, custom SOAs that are authored for BOMLines may not be directly reused in Active Workspace.

The customizers can use the **getBackingObjects** helper JavaScript API to access the backing BOMLine object for BVR structures. Once the backing BOMLine object is accessed, customizers can invoke any existing custom SOAs to make updates to the BOMLines.

The **getBackingObjects** helper JavaScript API accepts an array of **Awb0Elements** as input and returns BOMLines in the same order. This API works for non-indexed BVR structures only and for any other inputs, the output is null.

The **getBackingObjects** API is an asynchronous API and is used in the batch action declarative style. The batch actions could be configured to pass data from one action to another. These actions are executed in the exact sequence that is defined in the configuration.

> Note:
>
> When a custom SOA is executed, only *update* and *delete* actions are reflected immediately in the Active Content view. These are visible if the corresponding **Awb0Elements** are visible in the currently loaded structure.

The following syntax illustrates how you can define a batch action in the `actions` object of **viewModel.json**:

```
"BatchActionName": {
        "actionType": "batchJob",
        "steps":[
                  {
                     "action":"NameOfAction1",
                     "outputArg":["Array of String"]
                  },
                  {
                       "action":"NameOfAction1",
                       "inputArg":{ Object of 'parameterName' :
'{{actionData.OutPutArgName}}' }
                       "condition":"conditions.nameOfCondition"
                  }
```

```
            ]
        }
```

In the code, **BatchActionName** is the batch action and **batchJob** is the new action type. **Steps** is an array of object such as:

- **action**: (Required) Name of the action from **viewModel.json**
- **outputArg**: (Optional) Array of string that is mapped with the outputData of an action in sequence and forwarded to the next action
- **inputArg**: (Optional) Pair value that is evaluated to parameters of action properties
- **condition**: (Optional) The action is executed based on the condition

Example of using the getBackingObjects helper JavaScript API

To update **Awb0Elements**, you can use the **getBackingObjects** API to run any existing custom SOAs that are authored to work on BOMLines.

For example, the customizer has authored a custom SOA called **customSOA()** to update or customize BOMLines. This SOA takes BOMLines as input. This SOA can be reused in Active Workspace after the helper API **getBackingObjects** returns the BOMLines corresponding to the **Awb0Elements**. Any updates to the BOMLine using the **customSOA()** are reflected in the corresponding **Awb0Element** in Active Workspace.

The following are the prerequisites for this use case:

- The customizer has a custom client module for client development.
- The customizer has a declarative custom command in Active Content view.

On executing the batch action shown in the following code, two actions are triggered sequentially.

1. The asynchronous API **getBackingObjects()** will get BOMLines corresponding to the **Awb0Element**.
2. The customSOA is called with BOMLines that are received from the previous action.

```
"customBatchAction": {
            "actionType": "batchJob",
            "steps": [ {
                    "action": "getBackingObjects"
                },
                {
                    "action": "customSOA"
                }
```

```
            ]
        },
        "getBackingObjects": {
            "actionType": "JSFunctionAsync",
            "method": "getBackingObjects",
            "deps": "js/occmgmtBackingObjectProviderService",
            "inputData": {
                "viewModelObjects": "{{ctx.mselected}}"
            },
            "outputData": {
                "bomLines": ""
            }
        },
        "customSOA": {
            "actionType": "TcSoaService",
            "serviceName": "ServiceNamne",
            "method": "customMethos",
            "inputData": {
                "elements": "{{data.bomLines}}"
            },
            "outputData": {
                "customSOAResponse": "",
            },
            "deps": "js/customService.js"
},
```

Add custom properties to BOM columns in the rich client

You can add custom properties to the BOM columns in the Structure Manager or other applications where BOM lines are displayed. Following is a typical set of circumstances that results in your needing to add custom properties to the BOM columns:

- You have added custom business objects to the Business Modeler IDE that are subclassed under the **ItemRevision** business object, and you have added custom properties to those custom business objects.
- You have decided instances of these custom business objects should be organized into structures, and you want the custom properties to be visible when looking at the structures.
- You have decided on an application to display the structure, such as Structure Manager, Systems Engineering, and Manufacturing Process Planner, among others.

A simpler way is to use the **Fnd0BOMLineRevConfigProps** global constant in the Business Modeler IDE to add the custom properties from the custom item revision business object.

You can also define compound properties directly on the **BOMLine** business object and add them to the columns. However, you cannot define compound properties directly on the **ImanItemLine** business object.

The **Fnd0BOMLineRevConfigProps** global constant behaves similarly to other global constants that control derived BOM line properties. For each, add your custom business object type to the constant, and the properties on the custom type are added as **bl_** properties on the **BOMLine** business object.

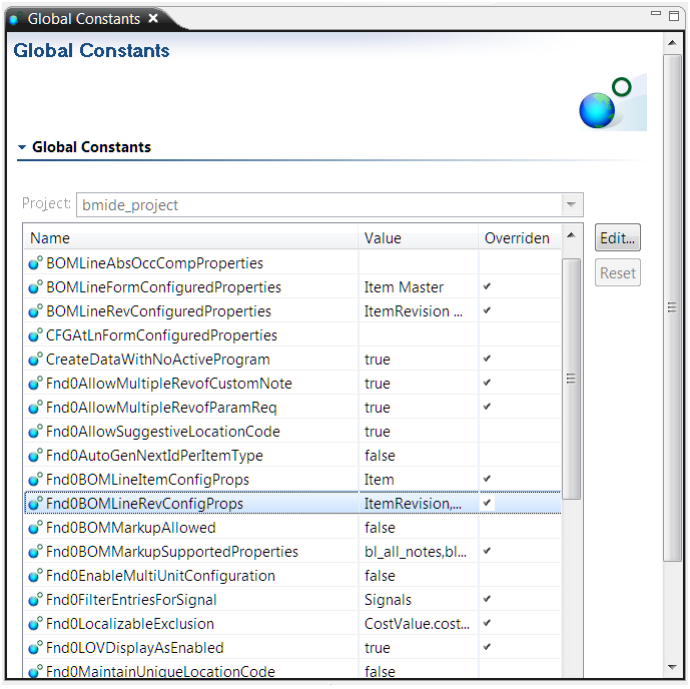
Property	Description
BOMLineFormConfiguredProperties	Adds properties from item master types.
BOMLineRevConfiguredProperties	Adds properties from item revision master types.
Fnd0BOMLineItemConfigProps	Adds properties from item types.
Fnd0BOMLineRevConfigProps	Adds properties from item revision types.

To add custom properties:

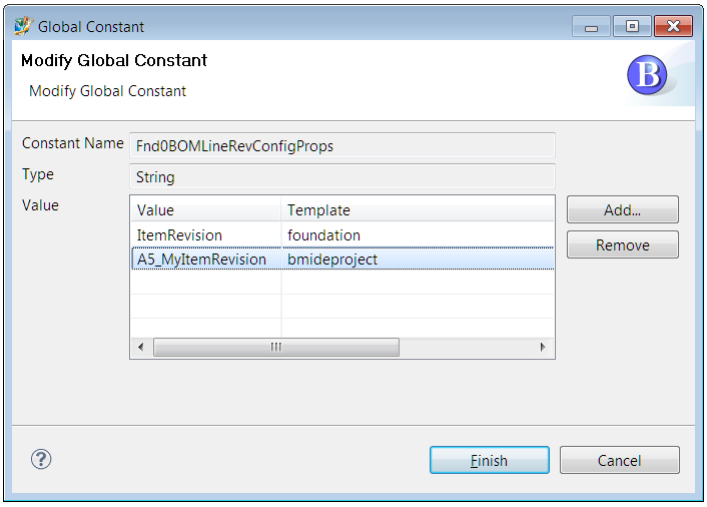
1. If you have not already done so, create a custom template project to hold your data model changes.

2. On the menu bar, choose **BMIDE→Editors→Global Constants Editor**.

3. Select the **Fnd0BOMLineRevConfigProps** global constant and click the **Edit** button.



4. In the **Modify Global Constant** dialog box, click the **Add** button, add a custom item revision business object, and click **Finish**.

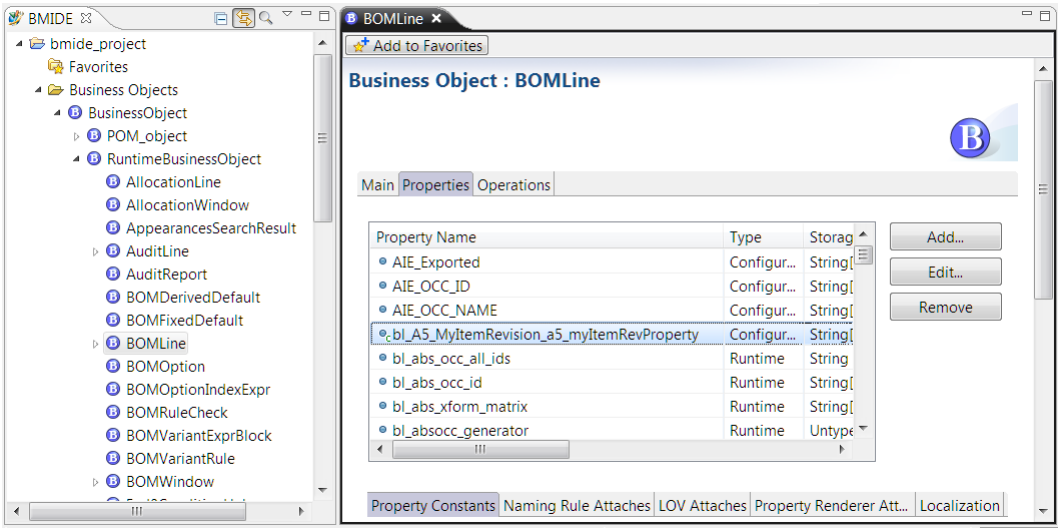


5. Right-click the project in the **BMIDE** view and choose **Reload Data Model**.

6. In the **Business Objects** folder, open the **BOMLine** business object and click the **Properties** tab.

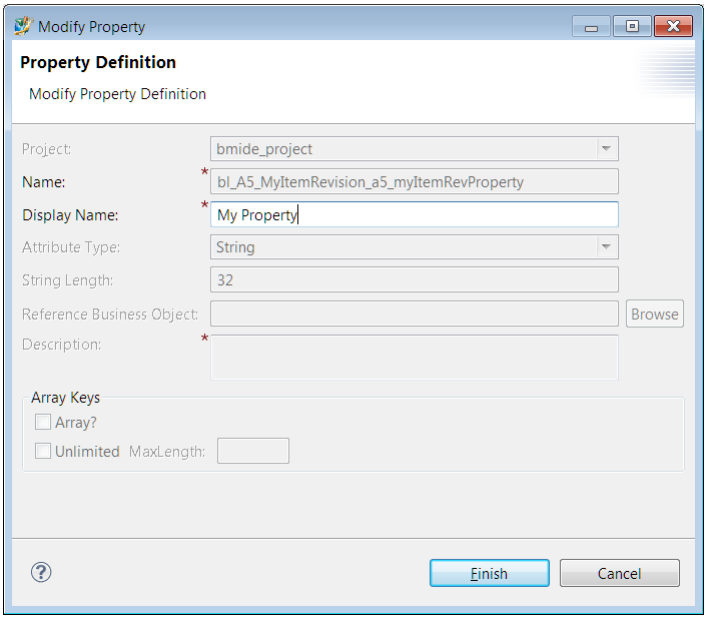
   In the **Properties** table, you see that the properties from the custom business object are added with the following naming convention:

   bl_*business-object-name_property-name*



7. Select a new **bl_** property, click the **Edit** button, and in the **Display Name** box, type the name you want to use for this property when it appears as a BOM column heading in the user interface (for example, **My Property**).

> Tip:
>
> If you do not perform this step, the property appears in the user interface with the internal property name.
>
> To avoid confusion and to make it clear that they are actually the same property, make the display name of this property the same as the display name on the source custom property.



8. Choose **BMIDE→Save Data Model** on the menu bar.

9. Deploy the template to a test server by choosing **BMIDE→Deploy Template** on the menu bar.

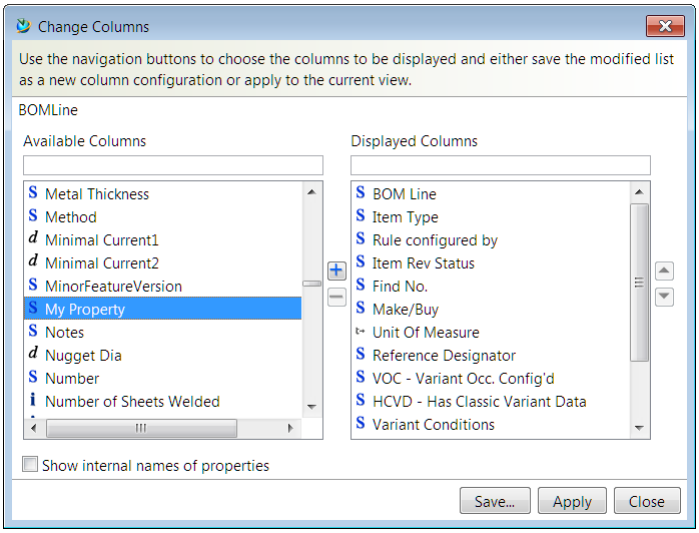
10. Perform the following steps to verify the new properties in the user interface.

    a. Run the rich client on the test server.

    b. Send an item revision to an application that has a BOM table, such as Structure Manager.

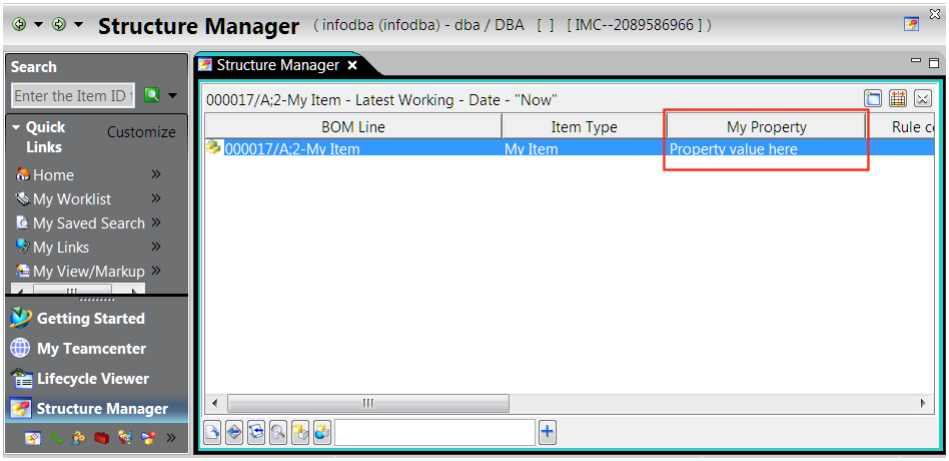
    c. Right-click a column heading in the BOM table and choose **Insert Column(s)**.

    d. In the **Available Columns** box, scroll down until you see your custom property.



e. Select the custom property and click the **+** button to add it to the **Displayed Columns** list. Then click **Apply**.

The new property displays a BOM table column.



Note:

You can use the **BOM_Properties_For_Column_Selection** preference to restrict the properties that are displayed on a column by group or role.

Restricting the properties displayed for column selection in the rich client

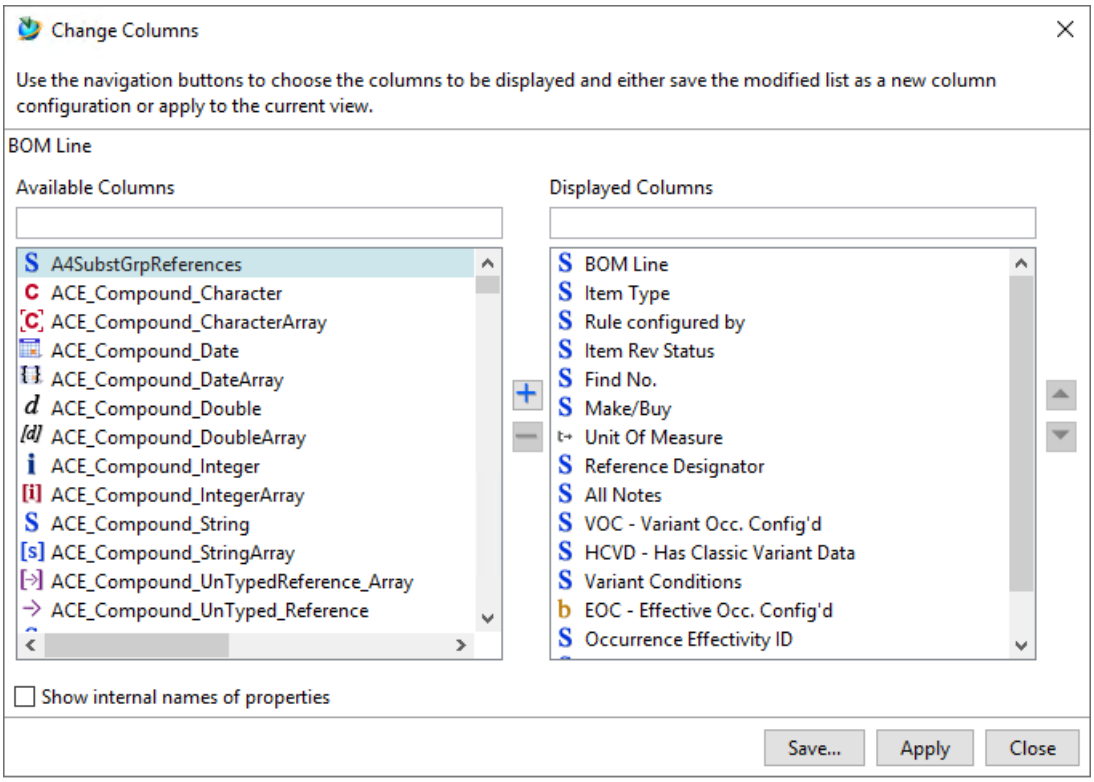
As an administrator, you can restrict the properties that are displayed for columns to be selected by a user. To do this, you can use the **BOM_Properties_For_Column_Selection** preference.

A user can add or remove property columns in a table tree by right-clicking the BOM window and then choosing **Insert Columns** in the shortcut menu. By default, Teamcenter displays all available properties to add or remove that may contain a large number of system and custom properties. To make the list more manageable and help the user to choose the required properties easily, you can restrict the properties that are displayed by group or role.

At the site-, group-, role-, and user-level, the **BOM_Properties_For_Column_Selection** preference can have different set of properties. Teamcenter chooses the intersection of the preference values of all levels.

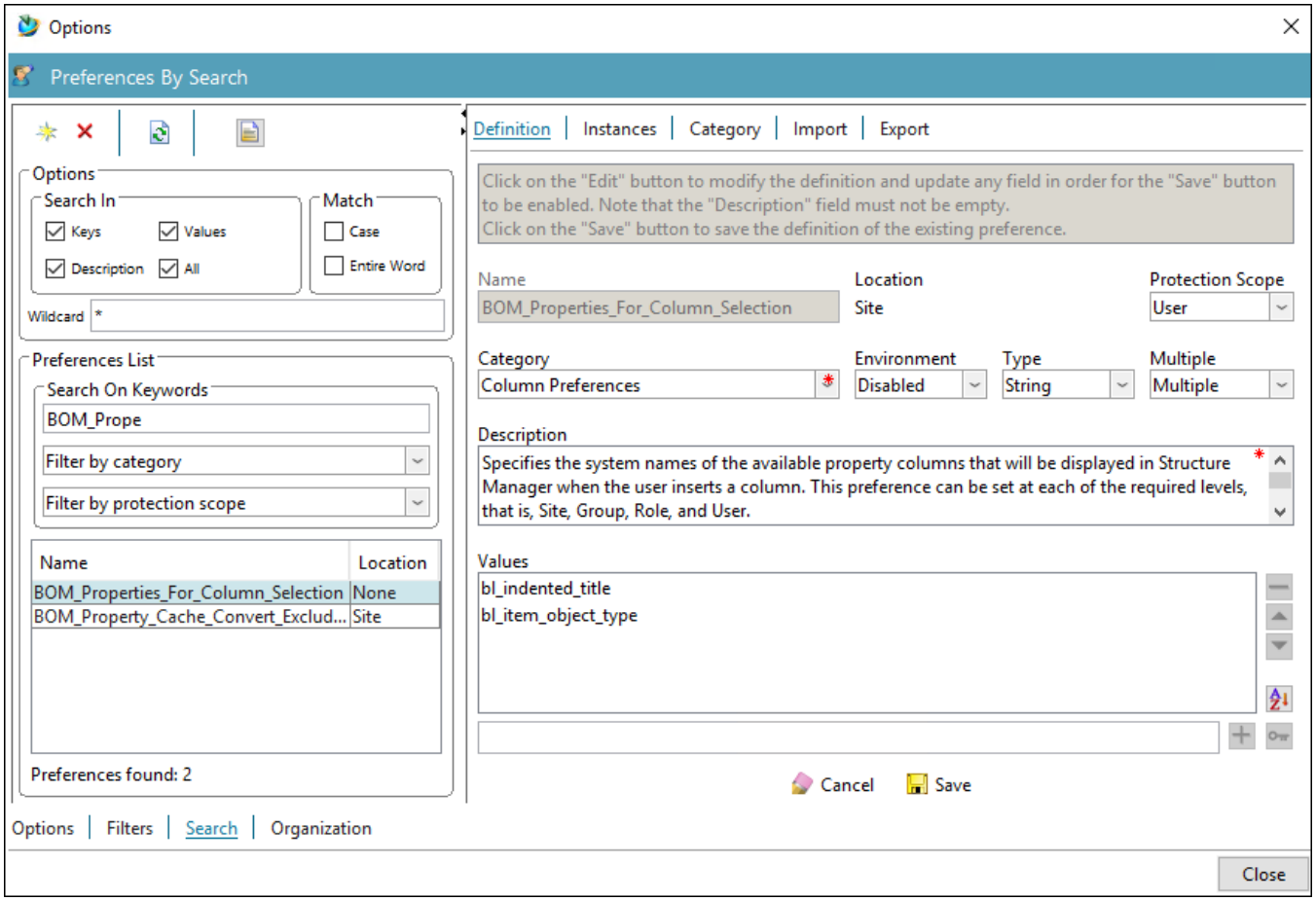
Example

In the **Displayed Columns** section, the user can see a few columns displayed, by default. However, in the **Available Columns** section, hundreds of columns are displayed.

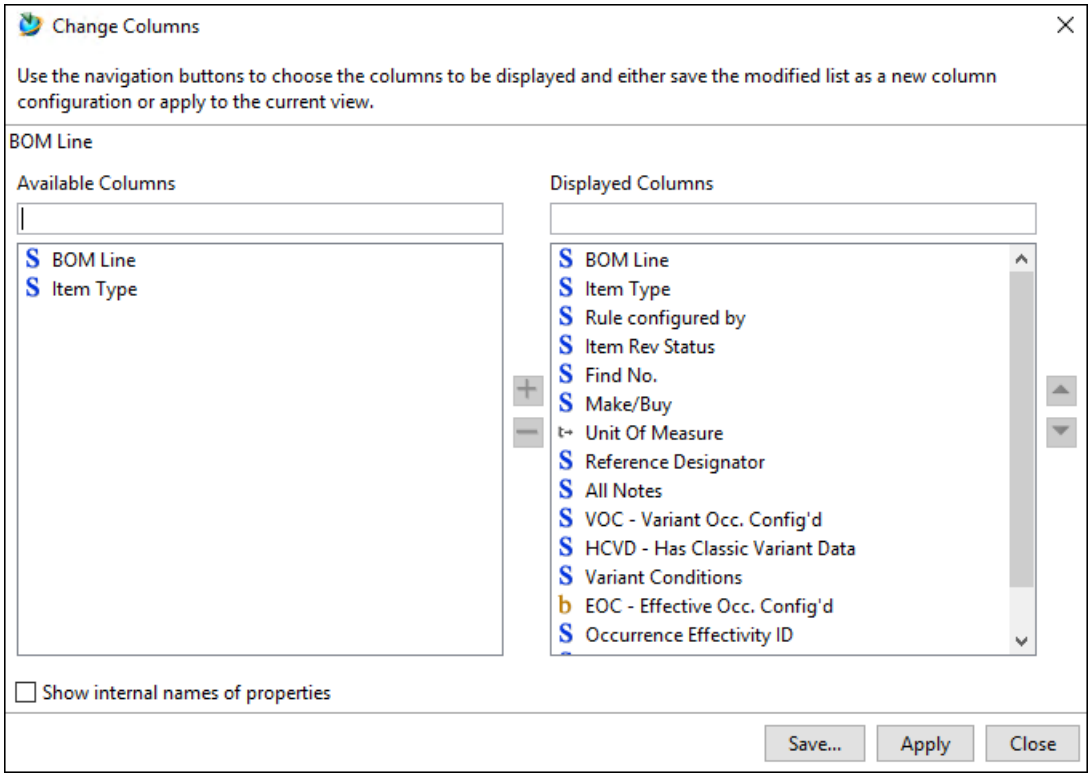


You want to make only some columns available for the users to work on. So, you use the **BOM_Properties_For_Column_Selection** preference to specify only those columns that you want the users to see in the **Available Columns** section.

You specify a couple of property values for the preference. To see the internal names of properties, you select the **Show internal names of properties** check box.



Only the properties that you specified are shown in the **Available Columns** section.



Enable the display of custom objects set to manage structures on Active Workspace

Add custom objects to the Content tab and search

To display your custom business object in the **Content** tab of Active Workspace, you must add it to the **Awb0SupportsStructure** global constant in the Business Modeler IDE. This procedure also allows instances of the custom business object to be returned by in context searches if the object is indexed, and also adds the **Add Element** button for the custom object.

1. In the Business Modeler IDE, ensure the **Active Content Structure** template is loaded and then open the Global Constants Editor.

2. Select the **Awb0SupportsStructure** constant and then click **Edit**.

3. In the **Add** dialog box of the **Edit** window, enter the name of the class corresponding to the first custom object you want to display.

4. Repeat the previous step for each of the other custom objects you want to display.

5. Click **Finish** on both, the **Modify Global Constant** and the **Add Value** dialog boxes to save the additions.

> Note:
>
> On some systems, you may have to reopen each additional object in the **Edit** window and save the entry again for each added item. Failure to save your entries on the first attempt does not necessarily indicate they are incorrect.

6. To enable the custom objects to be returned by in context searches, set the **Awp0SearchIsIndexed** business object constant to **true** on each custom object.

Display the Content tab with custom business object types

If you have custom business object types that have assembly children, the **Content** tab will not be displayed by default for them. To get the **Content** tab to appear in this case, you must perform the following steps:

1. **Ensure that the corresponding custom business object type is listed** under the **Awb0SupportsStructure** global business constant.

2. Add the following page pertaining to the **Content** tab to the **ShowObjectLocation Summary** XRT.

```
<inject type="dataset" src="Awb0ContentTab"/>
```

3. If the **ShowObjectLocation Summary** XRT is not available, add a custom summary XRT for the required business object. Then, add the preference for **Custom_BO_type.showObjectLocation.<***SubLocation***>**. This maps the summary XRT to the business object type.

Creating custom supersedure forms to track the replacement history of parts

Create a custom supersedure form

You can associate a custom form with a supersedure and use the form to track supersedure attributes, such as effectivity or interchangeability. Your site must first assign a particular form to use with supersedures and enable the custom form functionality. Once the functionality has been enabled, you can associate the form with any supersedure. Only one form can be associated with supersedures.

Enable custom forms

To enable the custom form functionality, you must change a change management default property so that the **Create Form** button  appears in Structure Manager.

1. In the rich client root directory, find the **com\teamcenter\rac\ecmanagement\ecmanagement_user.properties** file.

2. In this file, add the **enableSupercedureForm** property and ensure it is set to **true**.

3. Set the value of the **supercedureFormType** property to the name of the form type to use as the supersedure form.

   The custom form button now appears in the vertical toolbar of the **Supersedure** pane.

   When a user completes the form, it is saved and associated with the selected supersedure by the **Cm0BOMHasSupersedureForm** relation.

> Note:
>
> If you later need to determine the form that is associated with a supersedure, query the value of the untyped reference run-time property **custom_form_tag** for the **Supersedure** type.

Display custom icons for business objects in the context search results

As an administrator, you can customize the display of different (custom) icons based on the property of the derived types **Awb0Element** or **Item Revision**. Once you have added custom icons, you must perform the following steps to ensure that these icons are displayed properly in the search results.

1. Modify **AwDataNavigatorViewModel.json** to include objects and properties in the **aceSearchPolicyOverride** section.

2. Modify **occMgmtPropertyPolicy.json** to include objects and properties used in icon customization.

Consider the following example where a custom icon **typeCustomIcon48** is added. The steps required to ensure that the custom icon is displayed properly in the search results are as follows:

For example, you add the following entry in **typeIconsRegistries** to **typeCustomIcon48** when **someCustomProperty** of the underlying object has the value **Test**. For more information about using the **typeIconsRegistry** file, see Registering icons (advanced) .

```
{
        "type": {
          "names": [
            "Awb0DesignElement"
          ],
          "prop": {
            "names": [
              "awb0UnderlyingObject"
            ],
            "type": {
              "names": ["ItemRevision"],
              "prop": {
```

```
                "names": ["someCustomProperty"],
                "iconFileName": "typeCustomIcon48",
                "conditions": {
                    "object_name": {
                      "$eq": "Test"
                    }
                }
            }
          }
        }
      },
      "priority": 11
    }
```

You must perform the following steps to display the custom icons added in the above example to the structure search results on Active Workspace.

Caution:

Before upgrading Active Workspace, you must back up the *AwDataNavigatorViewModel.json* and *occMgmtPropertyPolicy.json* files. And after upgrading Active Workspace, you must merge these files.

1. Modify **AwDataNavigatorViewModel.json** to include **ItemRevision** and **someCustomProperty** in the **aceSearchPolicyOverride** section.

```
"aceSearchPolicyOverride" : {
            "types": [ {
                "name": "Awb0Element",
                "properties": [ {
                        "name": "awp0ThumbnailImageTicket"
                    }, {
                        "name": "object_string"
                    }, {
                        "name": "awp0CellProperties"
                    },
                    {
                        "name": "awb0BreadcrumbAncestor",
                        "modifiers": [ {
                            "name": "withProperties",
                            "Value": "true"
                        } ]
                    },
                    {
                        "name": "awb0UnderlyingObject",
                        "modifiers": [ {
                            "name": "withProperties",
```

```
                    "Value": "true"
                } ]
            }
        ]
    }, {
        "name": "Fgd0DesignElement",
        "properties": [ {
            "name": "awb0UnderlyingObject",
            "modifiers": [ {
                "name": "withProperties",
                "Value": "true"
            } ]
        } ]
    }, {
        "name": "Cpd0DesignElement",
        "properties": [ {
            "name": "cpd0category"
        } ]
    }, {
        "name": "Wbs0ElementCondElement",
        "properties": [ {
                "name": "wbs0IsWorkElement"
            },
            {
                "name": "wbs0RevObjectType"
            } ]
    }, {
        "name": "ItemRevision",
        "properties": [ {
                "name": "someCustomProperty"
            } ]
    } ]
}
```

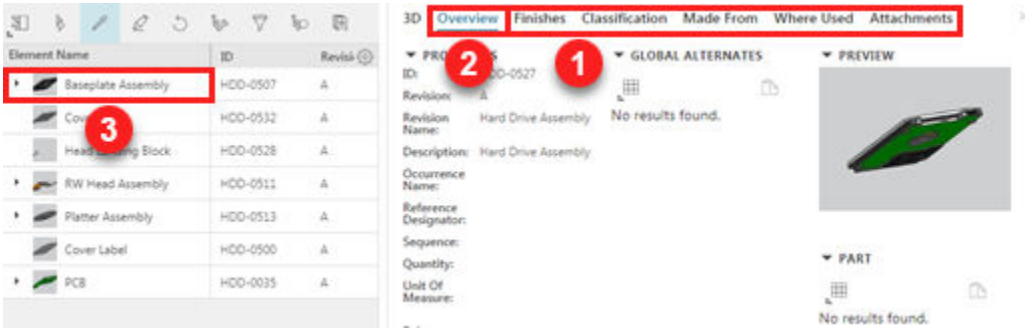
2. Modify **occMgmtPropertyPolicy.json** to include **awb0UnderlyingObject** with properties and the **ItemRevision** object with **someCustomProperty**.

4. Tasks to administer structures

5. Enable the display of structures on Active Workspace

Configure the Content tab

The **Content** tab displays the content of a structure. The following diagram shows the configurable parts of the **Content** tab.



Number	Element	How to configure
1	Occurrence sublocation tabs	Add the following preference: **AWC_***item-revision-type***.showObjectLocation.OccurrenceManagementSubLocation.SUMMARYRENDERING**. The value of the preference is the name of the style sheet dataset to take effect.
2	**Overview** tab	Use the summary style sheet of the associated element representing the occurrence. For example, use the **Awb0DesignElementSummary.xml** XML rendering style sheet file to configure the **Overview** tab for any design related item revision and use the **Arm0RequirementElementSummary.xml** file for requirement revisions.
3	Occurrence cell properties	Set the cell properties preference for the occurrence. Two examples are the **Awb0DesignElement.CellProperties** preference and the **Arm0RequirementElement.CellProperties** preference.

Tip:

To show the in-context search icon run the index on the BOM.

To view the **Architecture** tab, set the **Awb0AvailableFor** business object constant on the **Ase0ArchitectureFeature** business object in the Business Modeler IDE. Set the **Awb0AvailableFor** business object constant to list the business object types for which a feature should be made available, for example:

```
Functionality,Fnd0LogicalBlock,RequirementSpec,Requirement,Paragraph,Fnd0SystemModel
```

Modify the display name of the Content tab

You can modify only the display name of the **Content** tab by changing the value of **tc_xrt_Content** in the *activeworkspacebom_text_locale.xml* file. You must not change the **titlekey**.

Now you can update text server to override existing display names.

Control the visibility of commands in the Content tab

Because Active Content uses intermediary objects to represent their underlying objects in the **Content** tab, when you select one of these objects, you're actually selecting the intermediary object. Any declarative functionality, like conditions for example, are based on that selected intermediary object. However, there are many times when you will want the declarative client to base its decisions on the underlying object instead. Command visibility is the most common example of this.

Active Workspace provides the **AWC_AlternateSelectionCommandsList** preference to tell Active Content which commands must consider the underlying object instead of the intermediary object when determining their visibility. All commands listed by this preference will look to the *target object of the selected object* instead of the selected object itself when determining whether it is visible.

The shipped commands that are already available in the **Content** tab are already added to the preference. As an administrator, you can check them before adding new commands.

Add an LOV to a property in the Content tab

An LOV attached to the **BOMLine** property to which the **Awb0Element** property is mapped is not automatically displayed on the **Content** tab in Active Workspace. To display the LOV in the **Content** tab, you must attach the same LOV to the mapped **Awb0Element** property.

This applies to all children of the **Awb0Element** as well.

For more information, see *Attach an LOV to a property* in *BMIDE for Data Model Design*.

Choose properties to be displayed while adding structure elements

A user can create or update a structure on the **Content** tab in Active Workspace by adding new or existing structure elements to it. By default, while adding an element to a structure, users can specify **Quantity** in the **ELEMENT PROPERTIES** section. Depending on your site requirements, you can include additional element properties to be displayed on the **Add** panel in Active Workspace. To do so:

1. In BMIDE, locate the **PSOccurrence** business object.

2. In the **Operation Descriptor** > **CreateInput** tab, click **Add**.

3. In the **New Operation Input Property** dialog box:

   a. Select **Add a Property from Business Object** and click **Next**.

   b. Browse and add the required property in **Source Property**, for example, **ref_designator**.

4. Click **Finish**.

   Similarly, add other properties that you want to display in the **ELEMENT PROPERTIES** section.

After adding the required properties, you must update the **Awb0PSOccurrenceCreate.xml** style sheet to include these properties. For this:

1. Navigate to *TC_ROOT\install\activeworkspacebom\data* and open the **Awb0PSOccurrenceCreate.xml** file.

2. Add the newly added properties. For example, to add the **ref_designator** field:

```
<rendering>
    <page>
        <view>
            <property name="qty_value" />
            <property name="ref_designator" />
        </view>
    </page>
</rendering>
```

The element properties that you add are available for all business objects of type **Item** by default. If you do not want the properties to be available for certain business objects, add those objects as **Values** in the **AWBItemTypesToHideOccurrenceCreatePanel** preference.

By default, the **ELEMENT PROPERTIES** section is displayed in the **New** tab of the **Add** panel for the **Item** and **Part** business objects only. If you want this section to be displayed for other custom objects, you must update the create style sheet of those objects and append the following section:

```
<section>
          <inject type="dataset" src="Awb0ElementCreate"/>
</section>
```

Configure the properties of structured content

You can display the properties of the underlying object (called the **archetype**) in a structure element object tile. For example, you can show the release status of an item revision on the tile of an element in a structure using the following syntax: **awb0Archetype.release_status_list**.

You can use the home page of Active Workspace to provide tiles for your users' most commonly used pages, objects, and saved searches. For more information about defining object tiles, see *Organizing your users' common destinations* in *Active Workspace Customization*.

Specify the archetypes or underlying types that support creation of structures

The **Awb0SupportsStructure** global constant controls the types that can be opened in the active content explorer. This constant can take multiple values and each value is the name of a type that supports structure. Only the names of types that are specified in this constant can be opened in the explorer. The structure property is not inherited by subtypes, that is, you must add each subtype separately.

By default, this constant includes **ItemRevision**, **DesignRevision** and **PartRevision** as the values. For BVR structures, use the **ItemRevision** type as the value.

For more information, see *Change the value of a global constant* in *BMIDE for Data Model Design*.

Display the thumbnail instead of the icon for a structure element

Use the **AWB_ShowTypeIcon** preference to display either the thumbnail or the icon that indicates the type of a structure element.

The default value of the preference is **true**. In this case, the icon that indicates the type of the structure element is displayed everywhere except in the **PREVIEW** section. The icon is displayed even if the structure element has an attached thumbnail. For example, here, the item revision icon is displayed next to the Green Seat of Crosskart.

If you want to display the thumbnail for a structure element, set the value of the preference to **false**. When you do this, the thumbnail is displayed for the structure element if the structure element has a thumbnail attached to it. If the thumbnail is not attached, the icon is displayed. For example, here, a thumbnail showing a green seat is displayed next to the Green Seat of Crosskart.

Enable the display of connections and item elements

You can enable the display of connections (for example, welds) and item elements (for example, ports) in the tree structure and the 3D viewer in Active Workspace, using the following preferences:

Preference	Description
AWBShowConnections	Enables the display of connections, such as welds.
AWBShowPorts	Enables the display of item elements, such as ports.

The default value for these preferences is **false**. To enable the display, set the value for these preferences to **true**.

Prevent the complete expansion of a structure when it loads

By default, a structure is fully expanded when it loads. This is because the **BOM_closure_rule_name** preference does not have a value by default. To prevent the expansion of the complete structure, you can set a specific closure rule as the value of this preference.

For example, if you set the value of the preference to **BOMExpandSkipByItemType**, documents and requirements are not displayed in the structure.

Set the default quantity to 1 when the unit of measure is each

You can set the default quantity to 1 when the unit of measure for a structure element is *each*. In this case, when a business user adds a new element to a structure, the quantity is automatically set as 1 by default. Accordingly, when the unit of measure is not *each* by default, the quantity is not set as 1.

You can configure this using the **BOM_occ_default_qty_prevent_from_blank** preference. By default, the value for this preference is **false**. When you set the value to **true**, the default quantity for a newly added element is set to 1 when the unit of measure is each. However, when you set the value to **false**, the default quantity for a newly added element is not set to 1 even when the unit of measure is each.

After you update the preference value, the user does not see the quantity updated for the existing elements. The user can see the quantity set as 1 only for the newly added elements. To copy the value (of 1), the user must drag the quantity to the other elements in the structure.

Set how reference designators are displayed for packed lines

You can set how the reference designator values are displayed for the packed lines in a structure. You can do this using the **BOMRefDesignatorPackMode** preference.

When you set the value of the preference to **Range**, the reference designator values are collated into and displayed as ranges, for example, R1-R7 and D50-D100. However, when you set the value of the preference to **Concatenate**, all reference designator values of the packed lines are concatenated together with a separator. The business user views the reference designator values as a list with a separator, for example, P1, P2, P3, P4, and P5.

Set the separator for concatenating reference designators

You can set the separator for concatenating the reference designator values in a structure using the **BOMRefDesignatorPackSeparator** preference.

You can specify any character as a value for this preference. For example, if you specify a comma (**,**) as the value, a business user views the reference designators as R1, R2, R3, R4, R5.

Pack reference designators when packing structure elements

You can specify if the system must pack the reference designator values while packing structure elements. You can do this using the **BOM_Enable_Ref_Designator_Value_Packing** preference.

If you set the value of this preference to **true**, the system packs the reference designators. For example, consider that an element occurs six times in a structure. The reference designator values for these occurrences are C1, C2, C3, C6, C12, and C13. When a user packs this element, the reference designators too are packed. The **Ref Designator** column shows the values as C1-C3, C6, and C12-C13.

The user cannot edit the reference designators of packed lines. For the lines that are not packed, the user can modify the reference designator values irrespective of the value set for the preference.

Allow automatic update of quantity based on the number of reference designators

In a structure, you can set the quantity to be automatically updated as per the number of reference designators specified by a business user. To do this, you can use the **BOM_Enable_Quantity_Validation_Against_Ref_Designator** preference.

When you set the value of this preference to **true**, Teamcenter validates that the quantity and the reference designator list are consistent for every occurrence. In addition, when the reference designator list is modified by the user, the quantity is automatically set based on the number of reference designators specified by the user.

For this validation to work, you must also set the value of the **PS_Reference_Designator_Validation** preference to **true**.

Simplify the user interface by hiding the configuration parameters that the user does not need

By default, certain configuration parameters are always displayed in the configuration header and configuration panel in Active Workspace. You can choose to hide the configuration parameters that you do not need from view. To do this, use the **AWBEnabledStructureFeatures_Item** preference.

This preference is based on the type of the top-level element in a structure. By default, this preference is available for all item types. You can define a different configuration for any child type. For example, if you have a part or a design or a requirement as the top-level element in your structure, you can create the preference **AWBEnabledStructureFeatures_Part** or **AWBEnabledStructureFeatures_Design** or **AWBEnabledStructureFeatures_Requirement**, respectively.

From the preference, you can remove the values for the configuration parameters that you do not need. For the values that you remove, the corresponding configuration parameters are removed from the configuration header and configuration panel. In the menus on the user interface, the corresponding commands for the configuration parameters are made unavailable.

Based on your need, choose to remove a value from the relevant preference as follows:

To hide	Remove
The *unit effectivity* in the configuration header and the configuration panel.	The value **Awb0UnitEffectivityConfigFeature** from the preference.
The *variant* in the configuration header and the configuration panel.	The value **Awb0VariantFeature** from the preference.
The *expansion rule (closure rule)* in the configuration header and the configuration panel.	The value **Awb0ClosureRuleFeature** from the preference.

Note:

To prevent the business user from authoring *date effectivity*, you can remove the value **Awb0DateEffectivityConfigFeature** from the preference. Even if you remove this value from the preference, date effectivity is always displayed in the configuration header and the configuration panel.

Specify the maximum number of results to be displayed in the Used in Structures section

A business user can view an item revision across all assemblies up to the top level in the **Used in Structures** section on the **Where Used** tab. As a system administrator, you can specify the maximum number of results that can be displayed in this section using the **TC_WhereUsed_Display_Limit** preference.

By default, a maximum of 50,000 results are displayed in the **Used in Structures** section. If you change this value in the preference, for example, to 20,000, a maximum of 20,000 results are displayed. If the number of available results exceeds this threshold value, only the first page of the results is displayed. Active Workspace subsequently cancels the export process, and the system displays an error message.

If you specify a very high value, one that is greater than 50,000, Active Workspace might become unresponsive and not display any results.

Specify if a user can update a snapshot with a different item revision

A snapshot captures the current state of the item revisions used in a structure. A snapshot folder contains a list of these snapshots as item revisions. When a user opens a snapshot folder, the structure displayed contains these specific revisions. The user can update a snapshot to use a different revision by replacing an item revision with a different revision. For the user to be able to do this, you must set the value of the preference **AWC_display_configured_revs_for_pwa** to **false**. Once set, the user can see the new item revision in the snapshot folder.

Configure the packing of structure elements with units of measure

You can configure how a structure element with a unit of measure is packed. To do this, you set the **BOM_Additional_Packing_Criteria** preference with the value **bl_uom**.

In this case, the structure elements with different units of measure are packed separately. The elements with the same unit of measure are packed together as long as the **Quantity** is not specified as As Required (**A/R**) and the **Find Number** value is the same. The elements for which the **Quantity** is specified as **A/R** are not packed together.

Configure the display of the quantity value for a structure element

You can configure how the quantity value is displayed for a structure element by specifying a valid value in the **BOM_manage_quantity_display_value** preference:

> Note:
>
> This preference works only on the elements that are not packed.

- **False**: This is the default value. If this value is set in the preference and if the quantity is not already specified, the quantity is displayed as blank irrespective of what unit of measure is specified.
- **True**: If this value is set in the preference and if the quantity is not already specified, the unit of measure setting determines how the quantity is displayed:

  When the unit of measure is specified as **each**, the quantity is displayed as 1 and when it is not specified as **each**, the quantity is displayed as zero (0).

Configure the display of the reference designator value for a structure element

You can configure the display of the reference designator property and validate its format and value.

Procedure

1. For this, you can use the following preferences:

To do this	Set the value to the preference
To pack the product structure lines that include reference designators.	Set the **true** value to the **BOM_Enable_Ref_Designator_Value_Packing** preference. For example, eight BOM lines with the reference designators **C1, C5, C6, C7, C10, C14, C15, C16** will be packed to one BOM

To do this	Set the value to the preference
	line with the reference designator property **C1, C5-7, C10, C14-16**.
In packed mode, view the values in a range in the **Reference Designator** field.	Set the **Range** value to the **BOMRefDesignatorPackMode** preference. For example, C1-C5.
In packed mode, view the concatenated values in the **Reference Designator** field.	Set the **Concatenate** value to the **BOMRefDesignatorPackMode** preference. For example, C1,C2,C3,C4,C5.
Set the separator for the **Reference Designator** values in concatenate mode.	Set the **,** (comma) value to the **BOMRefDesignatorPackSeparator** preference to see them comma separated.
Validate whether the format and uniqueness of the reference designator values must be checked while summarizing.	Set the **PS_Reference_Designator_Validation** preference to **true** to validate the format and uniqueness of the reference generator values. For example, the format can be one or more uppercase letters followed by integer numbers like C7 or C18. The preference also checks if the values of reference generator are unique.
Validate whether the values of the reference designator are to be verified with the structure element quantity.	Set the **BOM_Enable_Quantity_Validation_Against_Ref_Designator** preference to **true**, so that the quantity of the structure element is checked against the number of reference designator values. For example, if there are four capacitors, the reference designator property must also contain 4 values.

Configure the child items in a structure with the parent variant conditions

You can configure whether a child item should be included or excluded from the structure using the variant condition of the parent item, without applying a variant rule. To do this, specify a valid value in the **PSM_nested_variant_solve_product_types** preference. In a nested structure with multiple parent items, the variant condition of the topmost parent item takes precedence in determining the inclusion of child items.

- **None**: If this value is set in the preference, the child items are not configured in a structure using the variant condition of the parent items of no topmost item type.

- **All**: If this value is set in the preference, the child items are configured in a structure using the variant condition of the parent items for topmost item of all types.

- Enter a value of an item type for which you want the nested variant configuration to be available. The default value is set to **Fnd0AbsDgnProd**.

For example, suppose that the variant formula for a fuel tank assembly is set as weight equal to **V**. In such a case, all the child items of the fuel tank assembly whose variant formula does not contain to weight equal to **V** are automatically excluded from the structure.

Additionally, you can configure whether a child item should be included or excluded from the structure using both the variant condition of the parent items and the variant rules. To do this, specify a valid value in both the **PSM_nested_variant_solve_product_types** preference and **PSM_enable_nested_variant_solve_on_apply_svr_only** preference.

The following are the valid values for the **PSM_enable_nested_variant_solve_on_apply_svr_only** preference:

- **false**: This is the default value. If this value is set to the preference, the nested variant is configured with or without a variant rule applied on the structure.

- **true**: If this value is set to the preference, the child items are configured in a structure only if a variant rule is applied on the structure.

You can also view the variability conditions for all items in a structure in the **Net Variability** field. The **Net Variability** field shows the variant conditions for a child item by adding the variant condition of the parent items in a nested structure.

6. Configure product structure creation and updates

Create conditions to control permitted structure content

You can use the Business Modeler IDE to create conditions that define what types of content are allowed in a product structure. The following types of restrictions can be defined by conditions:

- Only certain **item types can be children or parents** of other types.
- **Properties of the parent or child object** must satisfy specified values or be NULL.

After you create the conditions, create verification rules to assign them to specific business object types and their children using the **Fnd0OccurrenceConditionValidation Teamcenter Component** object.

After you install the template containing the conditions to a server, end users cannot add content that is prohibited by the defined conditions. In Structure Manager, end users can validate the product structure against the conditions by selecting an item in a product structure and choosing **Tools→Validate Occurrences**.

Keep the following in mind when you create these conditions:

- If there are no conditions in the system that control occurrence structures, only those structures that follow the existing parent-child inheritance already in the system are permitted.
- When you create conditions, children business objects inherit conditions from their parents.
- If a user attempts to place an object into a structure that does not have a matching condition rule, it fails. For example, if a user attempts to place a **Dataset** object as a child of an **Item** object, and no condition is set up to allow it, it fails because the **Dataset** business object is not a child of the **Item** business object. Therefore, you must create conditions for all likely parent-child combinations that you anticipate can occur in product structures at your organization.
- Siemens Digital Industries Software recommends that you create conditions that resolve to true for these most commonly used parent-child occurrence structures, **Item-Item** and **Item-GDE**. This avoids validation failures that can occur if there are no occurrence validation conditions in the system for these structures.

Control parent-child product structures

The following example shows how to create conditions that allow or prohibit an object instance to be placed as a child under another object instance in a product structure.

In this example, you first create an **ItemRevision-ItemRevision** condition that resolves to true (expression **1=1**) and an **ItemRevision-GeneralDesginElement** condition that resolves to true, and then create verification rules that add these conditions to the **Fnd0OccurrentConditionValidation Teamcenter Component** object.

> Tip:
>
> Siemens Digital Industries Software recommends that you create these kinds of conditions to allow all required parent-child product structures that users create.

The example also shows you how to change the conditions to resolve to false (expression **1=0**) to prevent these parent-child occurrence structures. This demonstrates how you can create a condition to prevent one object type from being a child of another object type.

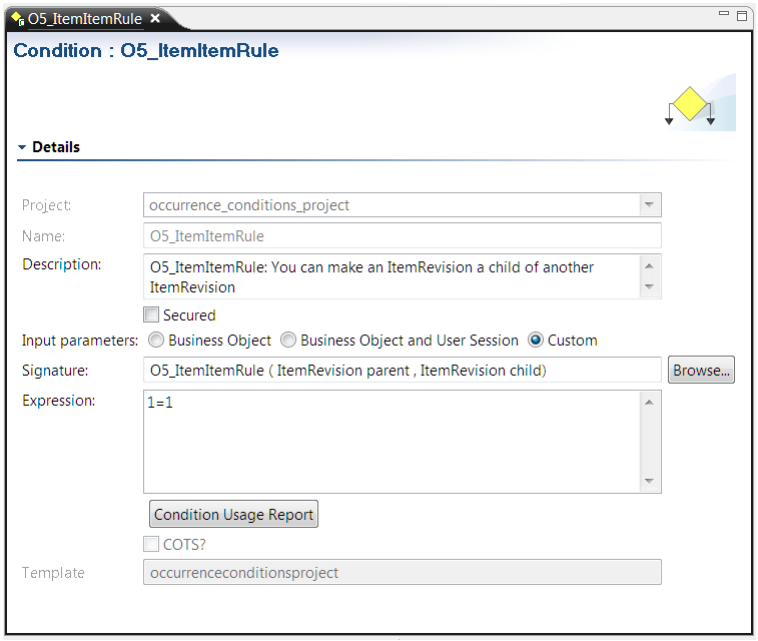
1. If you have not already done so, create a custom template project to hold your data model changes.

2. In the Business Modeler IDE, create the following conditions.

   > Note:
   >
   > In the following examples, **O5_** is the naming prefix associated with the template. When you create your own conditions, use the naming prefix associated with your template.

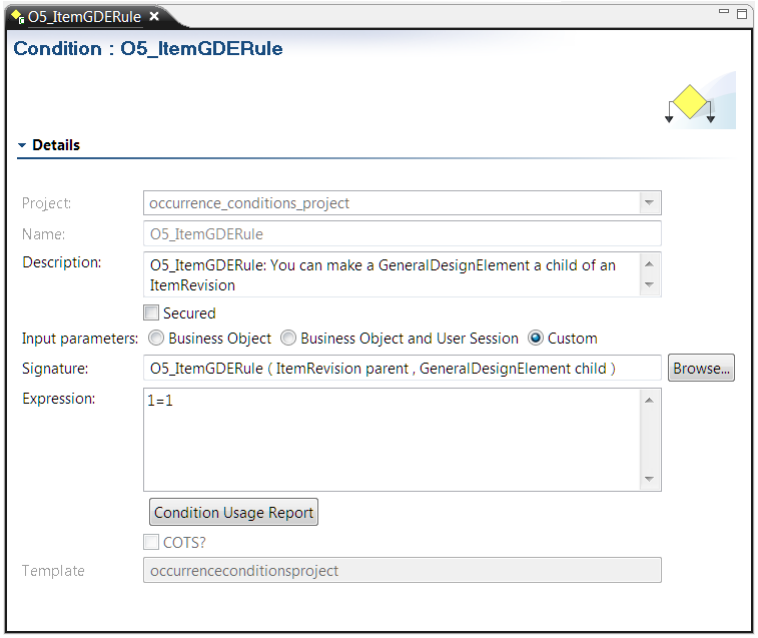
   - **O5_ItemItemRule**

     This condition allows users to add **ItemRevision** instances under other **ItemRevision** instances.



   - **O5_ItemGDERule**

This condition allows users to add **GeneralDesignElement** instances under **ItemRevision** instances.

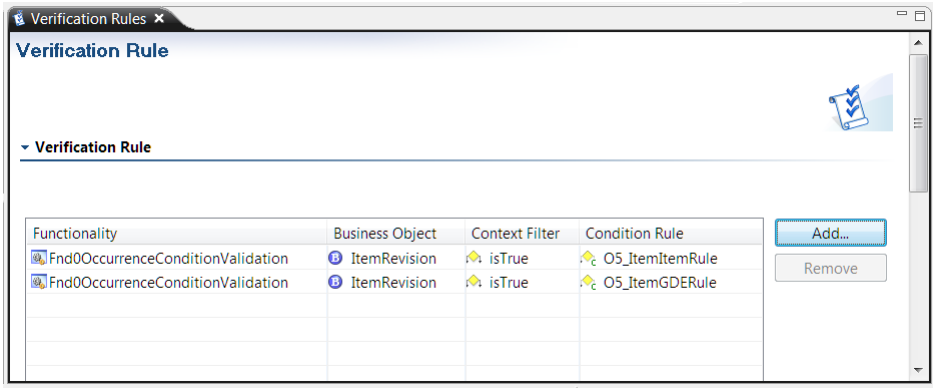


3. Create the verification rules.

   a. On the menu bar, choose **BMIDE→Editors→Verification Rules Editor**.

   b. Click the **Add** button in the **Verification Rule** tab and add verification rules as shown in the following figure. These rules use the **Fnd0OccurrenceConditionValidation Teamcenter Component** object, which assigns conditions to business object types for product structure validation.

> Note:
>
> The context filter for the verification rule must be set to **isTrue**. If multiple conditions are applied for the parent-child pair and at least one of them is matched, then it resolves to true.

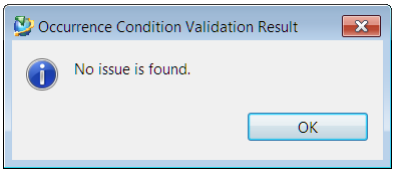


4. To save your changes to the template, on the menu bar, choose **BMIDE→Save Data Model**. Then choose **BMIDE→Generate Software Package** to package the template. Finally, use Teamcenter Environment Manager to install the packaged template to your server.

5. Verify the conditions.

   a. Log on to the rich client and open Structure Manager.

   b. To verify that you can add an item as a child under another item, create an item revision under another item revision. Then choose **Tools→Validate Occurrences**. You should receive the following message.



   c. Create a **GeneralDesignElement** instance under an item revision and repeat the validation step. It should also pass the occurrence validation.

> Note:
>
> To create **GeneralDesignElement** instances, choose **File→New→Item Element**. All available item elements are children of the **GeneralDesignElement** business object type, and therefore inherit the product structure conditions created for the **GeneralDesignElement** business object.

6. In the Business Modeler IDE, change the expressions on the conditions to resolve to false.

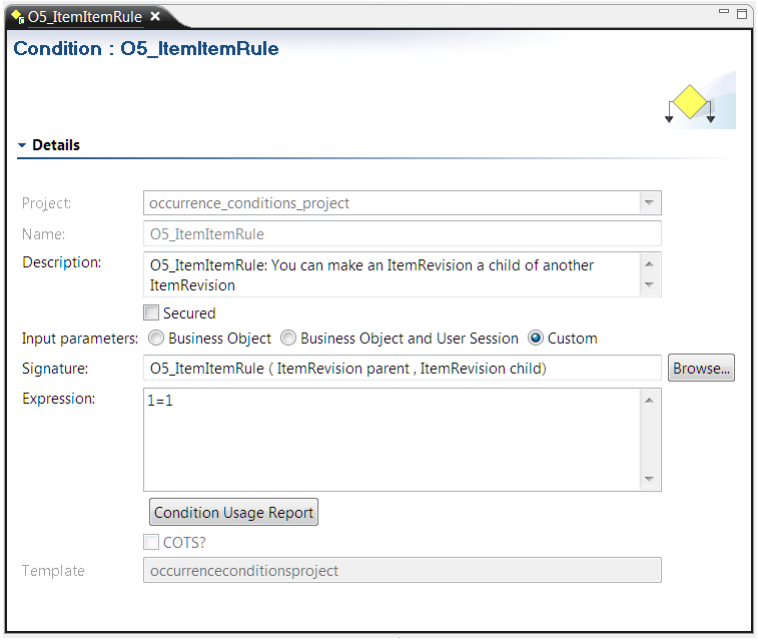
   - **O5_ItemItemRule**

     Change the **Expression** to **1=0** so that it resolves to false.

     Also change the **Description** to read:

```
O5_ItemItemRule: You cannot make an ItemRevision
a child of another ItemRevision
```

This text is included in the error message when occurrence validation fails when an **ItemRevision** instance is added under another **ItemRevision** instance.



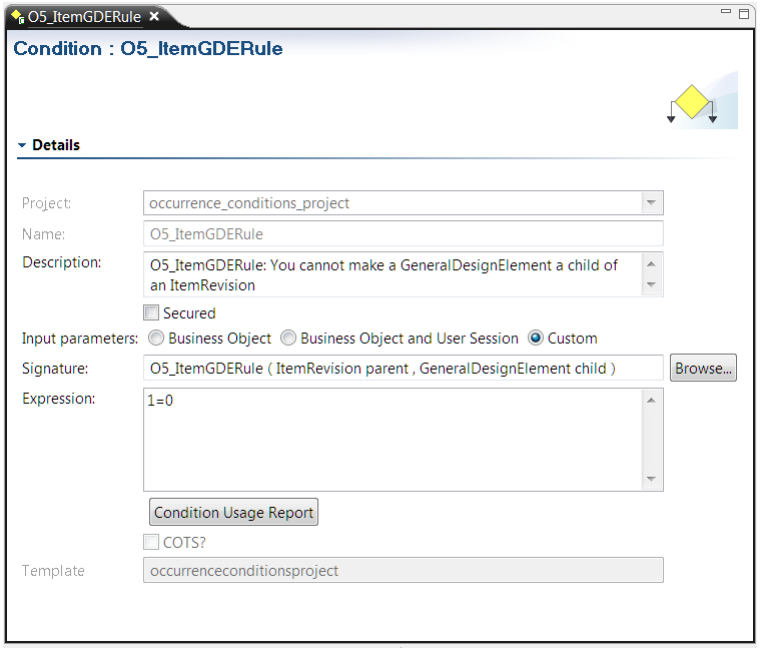
- **O5_ItemGDERule**

  Change the **Expression** to **1=0** so that it resolves to false.

  Also change the **Description** to read:

  ```
  O5_ItemGDERule: You cannot make a GeneralDesignElement
  a child of an ItemRevision
  ```

  This text is included in the error message when occurrence validation fails when a **GeneralDesignElement** instance is added under an **ItemRevision** instance.



7. Verify the conditions.

   a. Log on to the rich client and open Structure Manager.

   b. Create an item revision under another item revision, or select the structure you previously created (of an item revision under another item revision), and choose **Tools→Validate Occurrences**. You should receive an error message similar to the following:

```
The child object has failed the validation of the condition
"O5_ItemItemRule: You cannot make an ItemRevision a child of another
ItemRevision".
The occurrence condition validation has failed for the line
"000022/A;2-Another test item (View).
```

   The condition description text is incorporated into the error message. This verifies that the condition successfully prevents adding an item revision as a child of another item revision.

   c. Attempt to make a product structure with a general design element under an item revision, or select the structure you previously created. You receive an error message similar to the following:

```
The child object has failed the validation of the condition
"O5_ItemGDERule: You cannot make a GeneralDesignElement a child of an
ItemRevision".
The occurrence condition validation has failed for the line "test gde".
```

Now that you know how to create parent-child conditions to allow or prohibit certain occurrence structures, you can experiment on your own to create other conditions. This example is for illustration purposes only. You should create conditions that fit your business needs.

Control structures based on properties

The following example shows how to create conditions that allow or prohibit an object instance to be placed as a child under another object instance in a product structure based on an object's properties.

In this example, an item revision can only be placed as a child of another item revision if the child belongs to a specific project. First create a condition that resolves to true if the **project_ids** property equals a specific property, and then create a parent-child condition that points to the first condition.

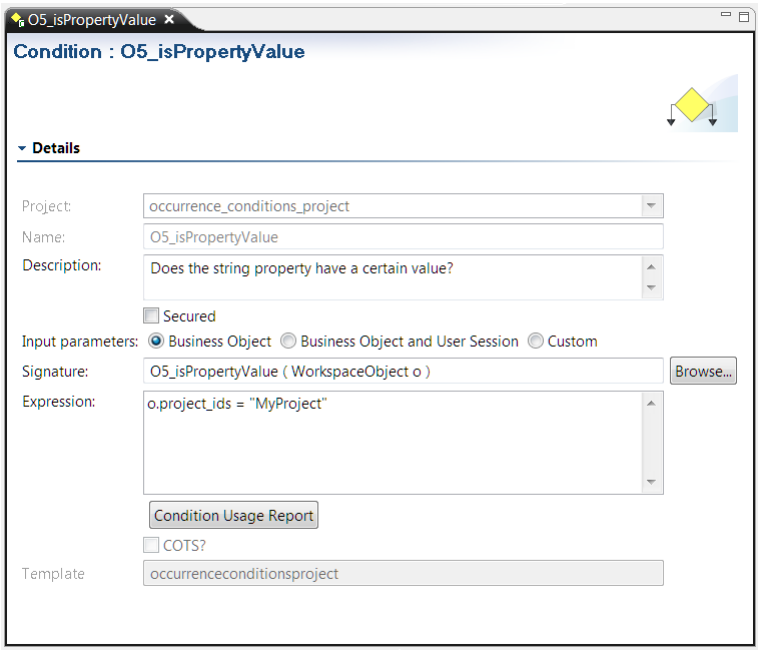
1. If you have not already done so, create a custom template project to hold your data model changes.

2. In the Business Modeler IDE, create the following conditions.

> Note:
>
> In the following examples, **O5_** is the naming prefix associated with the template. When you create your own conditions, use the naming prefix associated with your template.

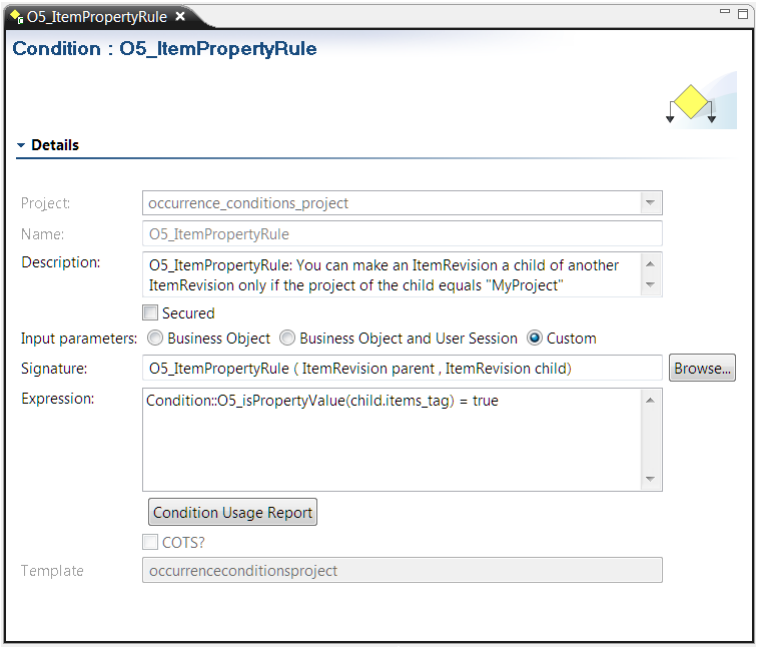
- **O5_isPropertyValue**

  This condition stipulates that the **project_ids** property must equal a certain value, in this case, **MyProject**.



- **O5_ItemPropertyRule**

  This condition allows users to add **ItemRevision** instances under other **ItemRevision** instances only if the **O5_isPropertyValue** condition resolves to true.

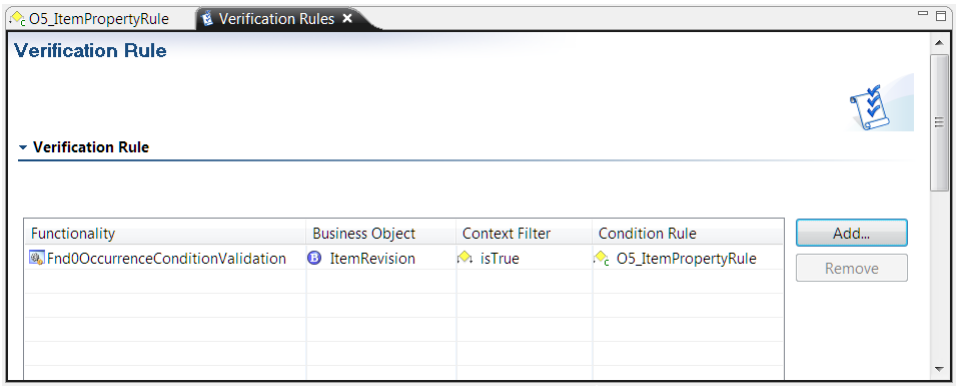


3. Create the verification rule.

   a. On the menu bar, choose **BMIDE→Editors→Verification Rules Editor**.

   b. Click the **Add** button on the **Verification Rule** tab and add a verification rule for the **O5_ItemPropertyRule** condition using the **Fnd0OccurrenceConditionValidation Teamcenter Component** object. The **Fnd0OccurrenceConditionValidation Teamcenter Component** object adds conditions to business object types for product structure validation.

> Note:
>
> The context filter for the verification rule must be set to **isTrue**. If multiple conditions are applied for the parent-child pair and at least one of them is matched, it resolves to true.



4. To save your changes to the template, on the menu bar, choose **BMIDE→Save Data Model**. Then choose **BMIDE→Generate Software Package** to package the template. Finally, use Teamcenter Environment Manager to install the packaged template to your server.

5. Verify the conditions.

   a. Log on to the rich client.

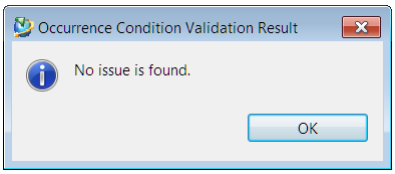
   b. In the Project application, create a project named **MyProject**.

   c. In My Teamcenter, right-click an item (not an item revision) and choose **Project→Assign** to assign it to the **MyProject** project. The item revision under the item is automatically assigned to this project.

   > Tip:
   >
   > To verify that the project is set correctly, you can view the object's **Project IDs** property.

   d. In Structure Manager, paste the item revision with the **MyProject** project assignment as a child under another item revision.

   To verify the occurrence validation, select the structure and choose **Tools→Validate Occurrences**. You receive the following message.



   e. Right-click the child item revision (with the **MyProject** project assignment), and choose **Project→Remove** to remove the **MyProject** assignment.

   f. Select the product structure and choose **Tools→Validate Occurrences**. You receive an error message similar to the following:

```
The child object has failed the validation of the condition
"O5_ItemPropertyRule: You can make an ItemRevision a child of
another
ItemRevision only if the project of the child equals
"MyProject"".
The occurrence condition validation has failed for the line
"000022/A;4-Another test item (View)".
```

   This verifies that an object can only be added as a child in the product structure if one of its properties has a specific value.

Now that you know how to create conditions to allow or prohibit certain occurrence structures based on property values, you can experiment on your own to create other conditions. Remember that this example is for illustration purposes only. You should create conditions that fit your business needs.

6. In the Business Modeler IDE, change the expressions on the conditions to resolve to false.

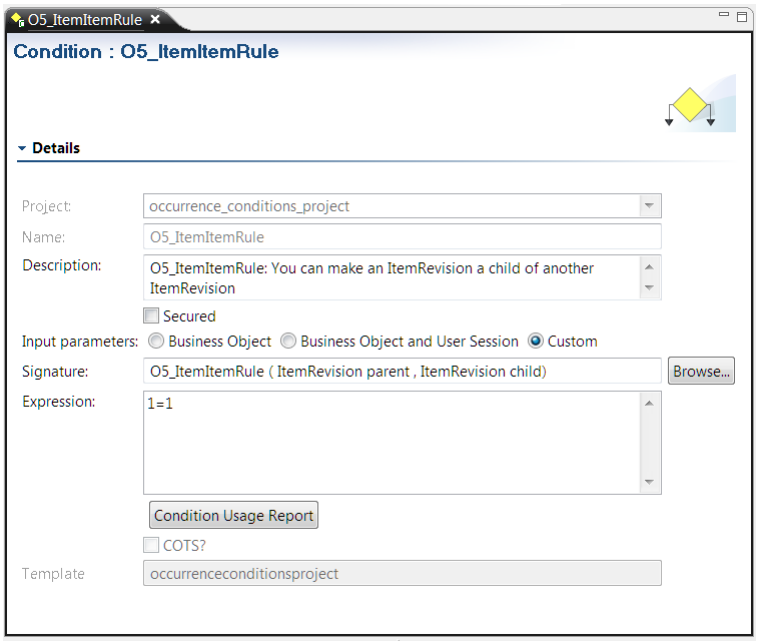
- **O5_ItemItemRule**

  Change the **Expression** to **1=0** so that it resolves to false.

  Also change the **Description** to read:

  ```
  O5_ItemItemRule: You cannot make an ItemRevision
  a child of another ItemRevision
  ```

  This text is included in the error message when occurrence validation fails when an **ItemRevision** instance is added under another **ItemRevision** instance.



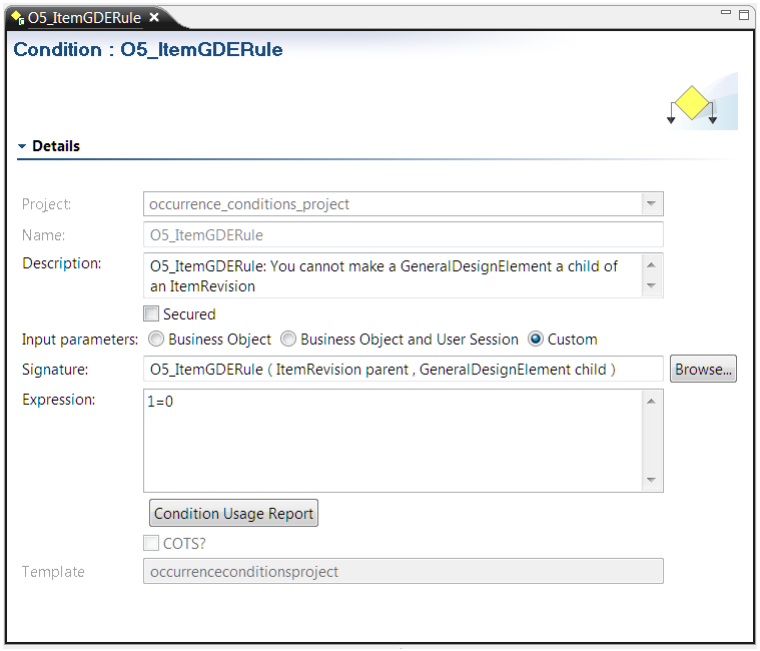
- **O5_ItemGDERule**

  Change the **Expression** to **1=0** so that it resolves to false.

  Also change the **Description** to read:

  ```
  O5_ItemGDERule: You cannot make a GeneralDesignElement
  a child of an ItemRevision
  ```

This text is included in the error message when occurrence validation fails when a **GeneralDesignElement** instance is added under an **ItemRevision** instance.



Configuring the duplication (cloning) of structures

A structure is duplicated (cloned) using the **Duplicate** command. The cloning action is either executed at the **Item Revision** level or the **Occurrence** level. The site administrator must set the **Structureless** preference to configure the duplication behavior at either the item revision level or at the occurrence level. The default value of the preference is *False* and this implies item revision level duplicate.

```
<preference name="AWBUseOccurrenceLevelStructureClone" type="Logical"
array="false" disabled="false" protectionScope="Site" envEnabled="false">
<preference_description>
Defines if the structure clone operation should be executed at
occurrence
level.
Occurrence level structure cloning is supported from platform TC12.3
onwards.
If this is set to true and platform supports Occurrence level clone
execution,
structure clone operation is executed at Occurrence level.
If this is set to true and platfrom does not support Occurrence level
clone
execution, clone operation is executed at Item Revision level.
If this is set to false, structure clone operation is executed at Item
Revision level.
</preference_description>
<context name="Teamcenter">
```

```
<value>false</value>
</context>
</preference>
```

You can customize the duplication operation by implementing the following user exits to:

- Override the duplication for certain BOM lines, for example, standard parts, using the following preprocess user exit.

```
/**
Gets the operation for a BOMLine during structure clone.
<br/>If @p use_default_operation is true, @p duplicate_operation will
be ignored.
Default opeartion configured by the system will be used.
<br/>If the user exit is not overridden, default opeartion configured
by the system will be used.

The following are the valid operations for @p duplicate_operation.
<ul>
<li>#STRUCTURE_CLONE_OPERATION_CLONE
<li>#STRUCTURE_CLONE_OPERATION_REFERENCE
<li>#STRUCTURE_CLONE_OPERATION_REVISE
<li>#STRUCTURE_CLONE_OPERATION_REPLACE
<li>#STRUCTURE_CLONE_OPERATION_IGNORE
</ul>

@returns
<ul>
<li>#ITK_ok on success.
<li>#BOM_invalid_tag if the @p bomline is invalid.
</ul>
*/
extern TCCORE_API int USER_bom_clone_get_operation(
const tag_t     bomline,
/**< (I) BOMLine to get duplicate operation. */
bool*           use_default_operation,
/**< (O) Use default duplicate operation for bomline. */
int*            duplicate_operation
/**< (O) Duplicate operation for bomline. */
);
```

- Establish a post-process equivalence between the source and cloned lines using the following post-process user exit. This user exit is invoked only once after the duplication process is complete. It establishes a relationship between the source and the cloned structures.

```
/**
Processes cloned objects after clone operation is complete.
<br/>Customizers need to replace the base action for the user exit
```

```
BMF_USER_bomline_process_cloned_structure to address their business
needs.

The following are cloning type options for @p cloning_type.
<ul>
<li>#ITEM_REVISION_CLONE
<li>#BOMLINE_CLONE
</ul>

@returns
<ul>
<li>#ITK_ok on success.
<li>#BOM_invalid_tag if the @p cloned_top_item_rev is invalid.
</ul>
*/
extern BOM_API int USER_bomline_process_cloned_structure(
 const tag_t     cloned_top_item_rev,
/**< (I) Top Item Revision of the cloned structure. */
 const int cloning_type,
/**< (I) Cloning type for clone operation. Expected values are:
<ul>
<li>#ITEM_REVISION_CLONE
<li>#BOMLINE_CLONE
</ul>
*/
const int n_source_bom_lines,
/**< (I) Number of BOMLines to be cloned. */
const tag_t*    source_bom_lines,
/**< (I) n_source_bom_lines List of BOMLines to be cloned. */
const tag_t*    cloned_occurrences
/**< (I) n_source_bom_lines List of cloned PSOccurrences. */
```

Note:

The **USER_bom_clone_process_cloned_structure** post-processor exit does not work. Use the **USER_bomline_process_cloned_structure** post-process user exit for this.

Enable the display of red lines to indicate structure changes

With a change notice set as an active change, BOM modifications and properties are identified by redlines. Your users can review active or closed changes associated with any structure. To enable highlighting the changes with red strikethroughs or in italicized green text, as the system administrator, you must set the **AWC_Enable_RedLine_feature** preference to **TRUE**.

In addition to this, you can also set the following preferences:

To do this	Set the preference to the designated value
Review the changes for the **Reference Designator** and **Quantity** fields in the structure.	Set the value of the **CM_bomline_tracked_properties** preference, to **bl_ref_designator** and **bl_quantity**.

Configure advanced search options within a structure

Suppose that your Teamcenter setup has the Context Management User license and the structure is indexed using Smart Discovery Indexing. In such a case, you can allow users to use the advanced search options within a structure by setting the **AW_Discovery_Advanced_Filter** preference to **true**. If you set this preference to **true**, users can create multiple OR groups for filter criteria, choose a group to modify the filter criteria, and also add a NOT group to the filter expression.

If you set the preference to **false**, users can use the filter expression with NX.

7. Configure BOM line properties

Adding compound properties

Introduction to compound properties

A *compound property* on a business object (the *display* or *target* object) effectively adds some property defined on another object (the *source* object) to the display object so that the source property behaves on the display object as if it is defined on the display object class.

A compound property defines a path between the display object and source object consisting of one or more reference relations, GRM relations, or a combination of the two. The path is used to traverse between the display object and the source object. You can add compound properties to COTS and custom business objects.

> Note:
>
> While a compound property enables you to add a property to a business object without writing custom code or using runtime properties, a compound property is not a replacement for runtime property functionality.

Set a compound property on a BOM line

If you want to set a compound property on a BOM line object in Structure Manager that displays an in-context value or incremental change value, you must use the **BOMLineAbsOccCompProperties** global constant. You can also create a compound property directly on a BOM line object, but it does not display the value of in-context edits (that is, the absolute occurrence value) or incremental change edits of the source property.

1. Set a value on the **BOMLineAbsOccCompProperties** global constant to create a compound property for use on a BOM line, for example:

   ```
   FORM::IMAN_specification::BVRSyncInfo::PROPERTY::BVRSyncInfo::
   last_sync_date::Absocc Last Sync Date
   ```

   This sample compound property displays the value of the **last_sync_date** property; **Absocc Last Sync Date** is the display name of the property.

2. Save the data model and deploy to the rich client.

3. To verify the compound property, perform the following steps in the rich client:

   a. Create a simple structure and send it to Structure Manager.

   b. Right-click the top line and select **Set In Context**.

c. Click the **Show/Hide Data Panel** button in the toolbar.

d. Select the child object in the structure and choose **File→New→Form**. Do this in context mode because the expectation is to display it only when the parent is loaded.

   The form is created as an attachment to item revision with the specification relation. The **Context Line** column in the **Attachments** tab of the data pane shows the context of the created form.

e. Open the form and edit the attribute values.

f. In Structure Manager, right-click the column headers, choose **Insert Columns**, and select the property you added using the global constant. The display name is the last item in the constant, for example, **Absocc Last Sync Date**.

   The column is added, and the value for that property is displayed in the column.

Make compound properties visible on item revisions

You can configure attributes on the item master form, and then use compound property rules so that the attributes are visible on item revisions. To achieve this, you must define and set the following global constants in the **foundation** template of the Business Modeler IDE:

- **BOMLineFormConfiguredProperties**

  Adds the properties from the form types in the constant to the BOM line. This configuration point was provided by the **PSE_add_props_of_item_form_types** preference in previous versions.

- **BOMLineRevConfiguredProperties**

  Adds the properties from the revision form type to the BOM line. This configuration point was provided by the **PSE_add_props_of_rev_form_types** preference in previous versions.

If either constant is set, all the form properties from the form type are added to the BOM line. By default, the foundation template sets the values of the constants to the item master and item revision master, respectively.

You can then add the attribute to display as a column. The attribute name is shown in fully qualified format, for example, if your item master attribute name is **A_SPLM**, the column name is shown as **bl_Item_Master_A_SPLM**.

You can optionally change the displayed column name.

Add a compound property on a GDE line using the bl_line_object property

Use the **bl_line_object** property instead of the **bl_revision** property to create a compound property to display the underlying GDE object's property on the GDE line.

General design element (GDE) objects are nonrevisable objects. These objects can appear in Structure Manager as GDE lines, just as items are seen in Structure Manager as item lines. In Structure Manager terminology, both GDE lines and item lines are BOM lines. Currently, Structure Manager can display properties of these BOM lines. Some of these properties are specific to item lines and others to GDE lines, while some are common to both GDE lines and item lines.

BOM line properties that are specific to item lines are not be applicable for GDE lines. For example, some item revision-specific BOM line properties are specifically applicable only to item lines and are not applicable to GDE lines. Some of these have revision property-specific text in their names, and because GDEs are nonrevisable objects, these are to be used only for item lines.

Following are some examples of properties not applicable for GDE lines:

bl_revision
bl_revision_change
bl_sequence_no
bl_config_string

Creating display names for BOM line properties

About display names

BOM line properties have both display names and system names, and the system names are in the format **bl**_*xxx*. You should define a display name for a system name if you add a custom business object type to your system.

Define a display name for a BOM line using the **BOMLineFormConfiguredProperties** and **BOMLineRevFormConfiguredProperties** global constants and compound properties available on the **BOMLine** business object in the Business Modeler IDE.

Caution:

Siemens Digital Industries Software recommends that you do not include a dot (.) in the property name. Such names cannot be edited in Structure Manager.

Note:

In previous versions, the administrator could create or change display names by editing the **/textserver/xml** files in the **lang** folder, for example, **system_property_names_locale.xml**. Display

names are no longer stored in XML files, and you can now only create or edit them in the Business Modeler IDE.

Create a display name

1. In the Business Modeler IDE, open the **BOMLine** business object and click the **Properties** tab.
2. Select the property whose display name you want to create and scroll to the **Localization** section.
3. Click **Add**.

   The Business Modeler IDE displays the **Add or Modify Localization** dialog box.
4. Enter the display name in the **Value Localization** box, select the locale, and set the status to **Approved**.
5. Save the data model and deploy the template.

BOM line naming behavior

A BOM internally evaluates the BOM line name from the **DisplayName** business object constant on the **Item Revision** business object. However, the lightweight BOM (LWB) functionality determines the BOM line name from the **object_string** property of the **Fnd0ItemLineLite** business object as defined in the **Fnd0ItemLineLite_bl_line_name_expression** preference. The default value of this preference is similar to the naming convention used by regular BOM line.

The default value of the **Fnd0ItemLineLite_bl_line_name_expression** preference is as follows:

```
$bl_item_item_id+"/"+$bl_rev_item_revision_id+";"+$bl_rev_sequence_id+"-"+
$bl_rev_object_name
```

The lightweight BOM naming convention deviates from regular BOM only in cases of subtypes of **ItemRevision** business objects that change their **DisplayName** business object constant expression.

In the following example of two lines in a structure (one with **Item** type and the second with **Part** type), the **DisplayName** business object constant is changed on the **PartRevision** business object.

Item Type	Structure	Regular BOM line name	LWB BOM line name
Item	**ItemComp**	**0000BNL34/A;1-NAME** (View)	**0000BNL34/A;1-NAME** (View)

The **DisplayName** business object constant for this line is:

```
$bl_item_item_id+"/"+$bl_rev_item_revision_id+";"+$bl_rev_sequence_id+"-"+
$bl_rev_object_name
```

Item Type	Structure	Regular BOM line name	LWB BOM line name
Part	**PartComp**	**0000BNL34-NAME** (View)	**0000BNL34/A;1-NAME** (View)

The **DisplayName** business object constant for this line is changed from the default value to the following:

```
$bl_item_item_id+"-"+$bl_rev_object_name
```

Enable validation for find numbers

Teamcenter assigns a unique find number to each line in the product structure. They provide an additional identifier or label for organizing the items in a single-level structure relationship.

You can set the following site preferences to validate find numbers:

Site preference	Description
PS_new_seqno_mode	Determines how new find numbers are allocated when items are inserted into a BOM view or BOM view revision. The default value is **new**. When it is set to this value, every item is given a new find number within the current BOM view, starting with 10 and increasing by 10. This preference has three settings: • **New**: Every item is given a new find number within the current BOM view, starting with **10** and increasing by increments of 10. • **Existing**: If an item with the same identifier already exists in the BOM view, Teamcenter assigns the inserted item the same find number. If not, the item is assigned a new find number according to the default sequence. • **None**: No find number is allocated to items inserted into a BOM view; users can add their own find number later.

Site preference	Description
	Note: After a find number crosses the value 2147483647, for every new line added, the find number starts from 8 and is incremented by 1 instead of 10.
PS_Find_Number_Validation	Determines whether the system validates find numbers. If validation is enabled by setting this preference, an error displays when the find number is zero or is not unique within the parent structure. By default, it is set to **false**. When it is set to this value, the system does not validate the Find Number.
PS_Duplicate_FindNo_Update	Determines whether duplicate find numbers of the same item are updated. By default, it is set to **true**. When it is set to this value, duplicate find numbers of the same item are updated.

Change the start and increment values for a find number

You can configure how *find numbers* are generated for structure elements by changing their starting value and increment value.

To define the starting value to be applied while generating a find number for the first occurrence in a BOM View Revision (BVR), you must set the **TC_BOM_INITIAL_SEQUENCE_NUMBER** preference. The default value of this preference is **10**, and you can specify any positive integer as a valid value.

To define the default increment to be applied while generating a sequence number for a subsequent occurrence under a BVR, you must set the **TC_BOM_SEQUENCE_NUMBER_INCREMENT** preference. The default value of this preference is **10**, and you can specify any positive integer as a valid value.

Define units of measure

Unit of measure is an attribute of the item. In Structure Manager, the BOM engineer specifies the value of the **Quantity** occurrence property in the units of measure for the component item (for example, 1.5 L for an **Oil** item).

The BOM engineer can optionally specify a quantity for a structure line in a user-defined unit of measure. To enable this option, you must set the **Fnd0PSEQtyConversionDSName** and **Fnd0PSEEnableQtyConversionUOM** global constants and create an XML file specifying all UOM conversion rules that are valid at your site.

For more information about these global constants, see the *Global constants addendum* in *Configuring Your Business Data Model in BMIDE*.

8. Set up import and export of structures

Set the properties to uniquely identify an occurrence during a structure import

While a business user is importing a structure from Excel, you can include certain properties as values in the **AWC_Occ_Unique_Identifier** preference to uniquely identify an occurrence that appears multiple times in the structure. By default, this preference contains the **bl_ref_designator**, **bl_occurrence_name**, and **bl_sequence_no** values. Any additional properties that you include in this preference are used to uniquely identify each occurrence of an element.

To know more about preferences in Active Workspace, see *Where can I get a list of preferences?*

Enable the import of remote components of a structure

If you have assembly structures that are replicated at several sites with Multi-Site Collaboration, Teamcenter may prompt users whether to import missing components—that is components that are not yet replicated at your site. If you set the **PSE_prompt_for_remote_import** preference to **on**, Teamcenter prompts if remote items should be imported when you expand the structure. When Structure Manager displays a **Ready** message, all remote components are imported successfully. If this preference is set to **off**, Teamcenter imports missing components without prompting the user.

Configuring structure export from Active Workspace to NX

The administrator must set the following preferences for the site to set up the export to NX feature for Active Workspace:

- Set the **allow_bb_bcz_export_import** preference to allow Briefcase export to complete the export NX data operation. The following example shows the preference details:

```
<preference name="allow_bb_bcz_export_import" type="Logical" array="false"
disabled="false" protectionScope="Site" envEnabled="false">
<preference_description>Allows briefcase export during Export NX Data operation.</
preference_description>
        <context name="Teamcenter">
            <value>true</value>
         </context>
    </preference>
```

- Set the **AWN0NX_NX_UnmanagedSite** preference to define the site to which you want to export the structure. The following example shows the preference details:

```
<preference name="AWN0NX_NX_UnmanagedSite" type="String" array="false"
disabled="false" protectionScope="Site" envEnabled="false">
      <preference_description>Defines the name of unmanaged offline site for Export NX
Data.</preference_description>
      <context name="Teamcenter">
```

```
        <value></value>
      </context>
    </preference>
```

- Set the **AWN0NX_ExportNotificationsCleanupDays** preference to specify the maximum duration for which the notification message and associated Briefcase file are retained after an export. The following example shows the preference to set the duration for 15 days:

```
<preference name="AWN0NX_ExportNotificationsCleanupDays" type="Integer" array="false"
disabled="false" protectionScope="Site" envEnabled="false">
      <preference_description>
        Specifies the maximum age (in days) for a export notification message and
associated briefcase dataset,
        after which the export notification and associated briefcase datasets is
deleted.
        Valid values are 1 through 365. If value is set to more than 365 then datasets
older than 365 days will be deleted.
      </preference_description>
      <context name="Teamcenter">
        <value>15</value>
      </context>
    </preference>
```

- Set the **AWN0NX_export_exclude_file_types** preference to specify the file types to be included in the export when the **Export Associated Files** option is selected during export. This preference is set at an individual user level.

```
<preference name="AWN0NX_export_exclude_file_types" type="String" array="true"
disabled="false" protectionScope="User" envEnabled="false">
      <preference_description>Controls the export of physical file types during Export
NX Data operation.</preference_description>
      <context name="Teamcenter">
        <value>UGMASTER:qaf,tso</value>
        <value>UGPART:qaf,tso</value>
        <value>UGALTREP:qaf,tso</value>
      </context>
    </preference>
```

Import a structure with variants and attribute groups

You can import a structure with variants and attribute groups from a *CSV* file into Teamcenter, using the **csv2tcxml** utility. Refer to the *foundation_ReadMe.txt* in the *TC_DATA\csv2tcxml\tdata\Integration\foundation* directory for sample files and import commands.

Prerequisites

Ensure that you have set up the **csv2tcxml** utility as described inTeamcenter Data Exchange.

In addition, ensure that the variability data is available in the Teamcenter database.

Procedure

1. In the rich client, set the default value of the **opt_tcxml_import_post_action** option as **true** in the **SiteConsolidationImportDefault** transfer option set.

2. Add the attribute **fnd0VariantFormula** to the business object **PSOccurrence** by:

   - Using *local_override.xml*. For more information on how to use this file, refer to the *Extend the framework* section in the *csv2tcxml_user_guide.pdf* and *foundation_ReadMe.txt* files.

   - Adding an attribute by using BMIDE.

3. Create and import a structure with variants and attribute groups. You can refer to the following file for a sample template to create your data.

   - *TC_DATA\csv2tcxml\tdata\Integration\foundation\ Bomstructure.csv*

   Once the data is ready for the import, you can convert and import it. Refer to the file *csv2tcxml_quick_start.pdf* located in the *TC_DATA\csv2tcxml* directory for conversion and import commands.

Create a Microsoft Excel template for exporting product structures

To perform a simple, static export of the product structure and view the formatted results in Microsoft Excel, do the following:

1. Add a custom Excel template, as follows:

   a. Ensure you are logged on as a user with Teamcenter administration privileges.

   b. Create a new item of type **ExcelTemplate** in the Teamcenter administrator's account **Home→Requirements Management Templates→ExcelTemplates** folder.

   c. Add the **MSExcelX** dataset underneath the **ExcelTemplateRevision** object.

      Teamcenter uses the named reference of this dataset as your template for structure export.

   > Note:
   >
   > Choose appropriate naming conventions for item, dataset, and named reference.
   >
   > Bypass naming rules if required.

2. To include any custom form properties in the report, create a custom transfer mode and add it to the Excel template. The transfer mode comprises a custom closure rule and a custom property set.

a. In the PLM XML/TC XML Export Import Administration application, create a custom closure rule to export the item revision master form. For example:

Primary object class	Primary object	Secondary object class	Secondary object	Relation type	Related property or object	Action type
Class	*	*	**Item Version Master**	**PROPERTY**	**IMAN_master_form_rev**	**PROCESS + TRAVERSE**

b. Create a custom property set that defines the custom form properties you want to export. The property set should contain a line similar to the following example for each custom property you want to export.

Primary object class	Primary object	Relation type	Related property or object	Action type
Class	**Form**	**PROPERTY**	*custom_form_name*	**DO**

3. Before configuring the Excel template, test the custom transfer mode works by performing a PLM XML export.

   a. In Structure Manager, select an item revision that includes your custom form properties.

   b. Choose **Tools→Export→Object→PLMXML→***your custom transfer mode*.

      Teamcenter creates a PLM XML output file.

   c. Verify the output file holds the values of the custom properties.

4. Configure the Excel template by adding the necessary columns for the custom properties. logos, and hyperlinks. Refer to the sample files for examples.

5. By default, every level in the Excel template is displayed in a different row in the final result. To merge values for a single revision into one row, you must apply packing to the **ExcelTemplate** item type.

   a. Check out the item.

   b. Select the **apply_packing** value for the **Excel Template Rules** property and modify it as necessary.

   c. Check in the item.

6. Perform the structure export by choosing **Tools→Export→Objects to Excel→Use Excel Template** and selecting your custom Excel template.

9. Configure the baseline feature

Configure the baseline naming rule

To configure the baseline feature, you can:

Use the Business Modeler IDE to create the appropriate baseline naming rules for your site and attach them to the appropriate item revision types. You must also create a **Baseline** status type in the Business Modeler IDE.

You can use site preferences with the *Baseline_* prefix and the *_baseline* suffix to configure how Teamcenter interprets legacy transformation data.

For information about retrieving a list of preferences, see *Where can I get a list of preferences?* in *Teamcenter Preferences* on Support Center.

Configure smart baselines

By default, the **ITEM_smart_baseline** preference is set to **1**. When the **ITEM_smart_baseline** preference is set to **1**, and a user tries to create a smart baseline of an assembly. In such a case, the new revisions are created for the associated items whose work in progress revisions are modified since their last baseline was created.

If you set the **ITEM_smart_baseline** preference to **0**, the new revisions for the associated items are always created.

10. Create predefined occurrence note types

You can associate occurrence note types with an occurrence in the product structure. Users can specify a value for any note type that is defined for the site.

Some standard note types are supplied to allow synchronization of occurrence attributes with NX. These note types can be displayed but not deleted.

You can define lists of values (LOVs) and default values for occurrence notes.

To create a new occurrence note type:

1. Start the Business Modeler IDE and, in the Extensions view, select the project in which you want to create the new note type. Right-click the project and choose **Organize→Set active extension file**. Select the **options.xml** file as the file in which to save the new note type.

2. Expand the project and the **Options→Note** folders.

3. Right-click the **Note** folder and choose **New Note Type** from the shortcut menu.

   The New Note Type wizard runs.

4. Perform the following steps in the **Note Type** dialog box:

   > Note:
   >
   > The **Project** box defaults to the already selected project.

   a. In the **Name** box, enter the name you want to assign to the new note type.

   b. In the **Description** box, enter a description of the new note type.

   c. Select the **Attach Value List** check box if you want to attach a list of values (LOV) to the note.

   d. If you selected the **Attach Value List** check box, click the **Browse** button to the right of the **LOV** box to locate the list of values to attach to the note. Type an asterisk * in the **Find** dialog box to see all possible selections. Click the **Browse** button to the right of the **Default Value** box to choose the value from the list of values that you want to use for the note type.

   e. Click **Finish**.

      The new occurrence note type displays under the **Note** folder in the **Extensions** view.

5. To save the changes to the data model, chose **File→Save Data Model**.

11. Set up structure indexing by using Active Content Structure

Introduction to structure indexing

> Warning:
>
> Indexing structures by using the Active Content Structure indexing feature is soon going to be deprecated. It is recommended that you now start indexing structures by using Smart Discovery Indexing.

Structure indexing is part of the Active Content Structure feature that provides support for fast structure (BOM) navigation and in-context search capability leveraged by Active Workspace.

If you are using Massive Model Visualization (MMV), structure indexing must be set up. MMV also requires Visualization Data Server (on Windows or Linux). The Visualization Data Server uses the structure indexing infrastructure of Active Workspace to keep cached product structures up-to-date.

For any given product configuration, there are two indexes maintained:

- An index within the Teamcenter database that is used for structure modeling and navigation.
- An index within Solr used for in-context searches. (Structure data is stored within the same Solr instance as object data but within a separate collection.)

The Structure feature is made up of several components:

- Template

  Contains the model and schema, the **tcserver** libraries, and so on.

- Active Workspace client contributions

  Contains the client exposure of the server features.

- Indexer and translator additions

  Contains indexing framework support and additions.

Use the **bomindex_admin** utility and the **TcFTSIndexer** to manage BOM index data. The **runTcFTSIndexer** utility maintains both Teamcenter and Solr indexes. Structure indexing support is added into the TcFTSIndexer installation during installation. Active Workspace should already be installed and working.

All indexing in Active Workspace is orchestrated by a single TcFTSIndexer instance. This is the *index orchestrator* that performs orchestration functions for all types of indexing (object data, structure, and so on). It is expected that the TcFTSIndexer is run with the **-background** argument, and separate commands are issued to control the indexing operations for the different indexing types. A given indexing action can be executed once or executed on a given interval.

Structure indexing recommendations

Indexing structure content requires effort. Therefore, you must understand what content should be indexed for the most benefit, and the configurations of that content that are appropriate and useful for index-enabled interaction.

For example, suppose you have a dozen significant top-level structures representing product platforms. This is a clear example of content that would benefit from being indexed. Even if that content were no longer active within the engineering organization or were out of production, there may still be value to sustaining that definition if users would benefit from the information about those products.

To help you decide what you should index, following are some examples.

- Must index
  - Large structures using **Massive Model Visualization**
- Should likely index
  - Product platforms under development
  - Requirement structure for a strategic initiative
  - Mature product in manufacturing production
- Should be considered for indexing
  - Moderately sized structures sharing considerable reused content
  - Out-of-production products with ongoing service
- Unlikely to need an index
  - R&D concepts
  - Legacy products out of production
  - Small and simple structures

Structure indexing provides where-used capabilities in the top-level assembly contexts that are indexed and facet-based search filtering capability.

Structure indexing requirements for Massive Model Visualization

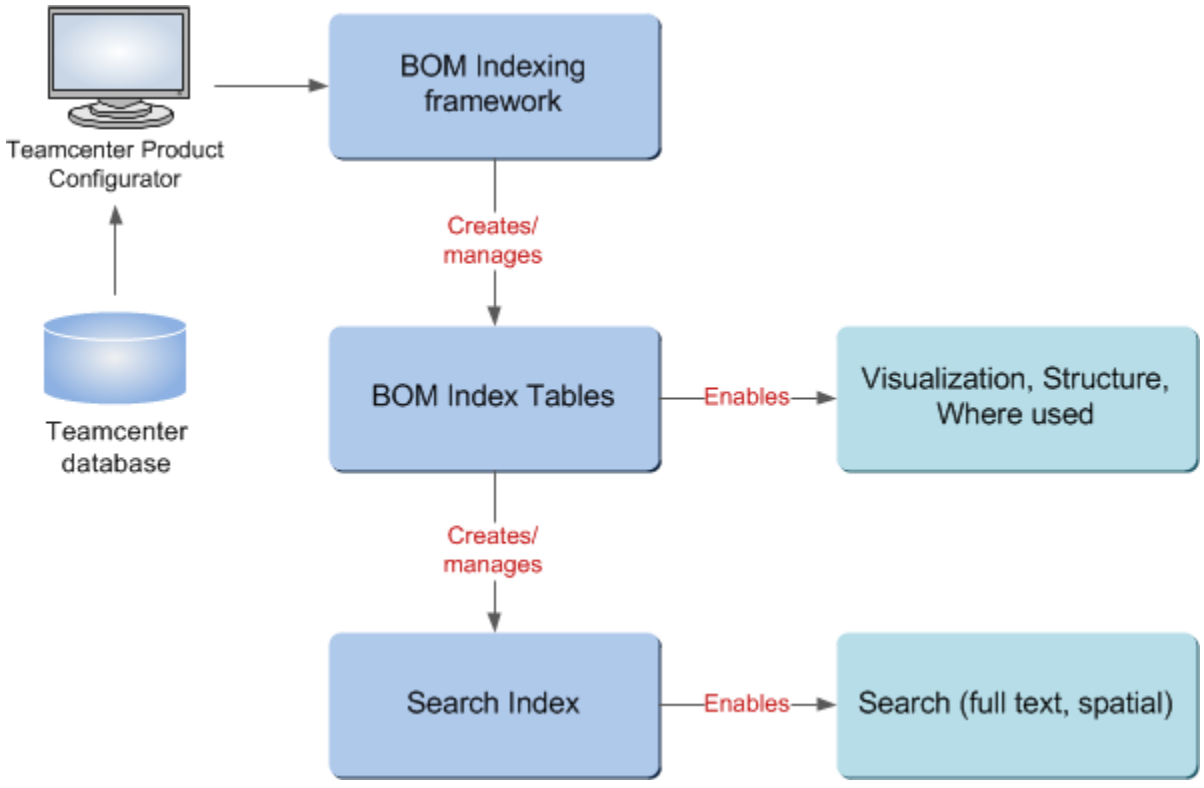
Structure indexing is required for Massive Model Visualization (MMV) large structures. MMV also requires Visualization Data Server (on Windows or Linux). The Visualization Data Server uses the structure indexing infrastructure of Active Workspace to keep cached product structures up-to-date.

You can retain interim files that represent changes between the last valid indexed version and the latest structure data.

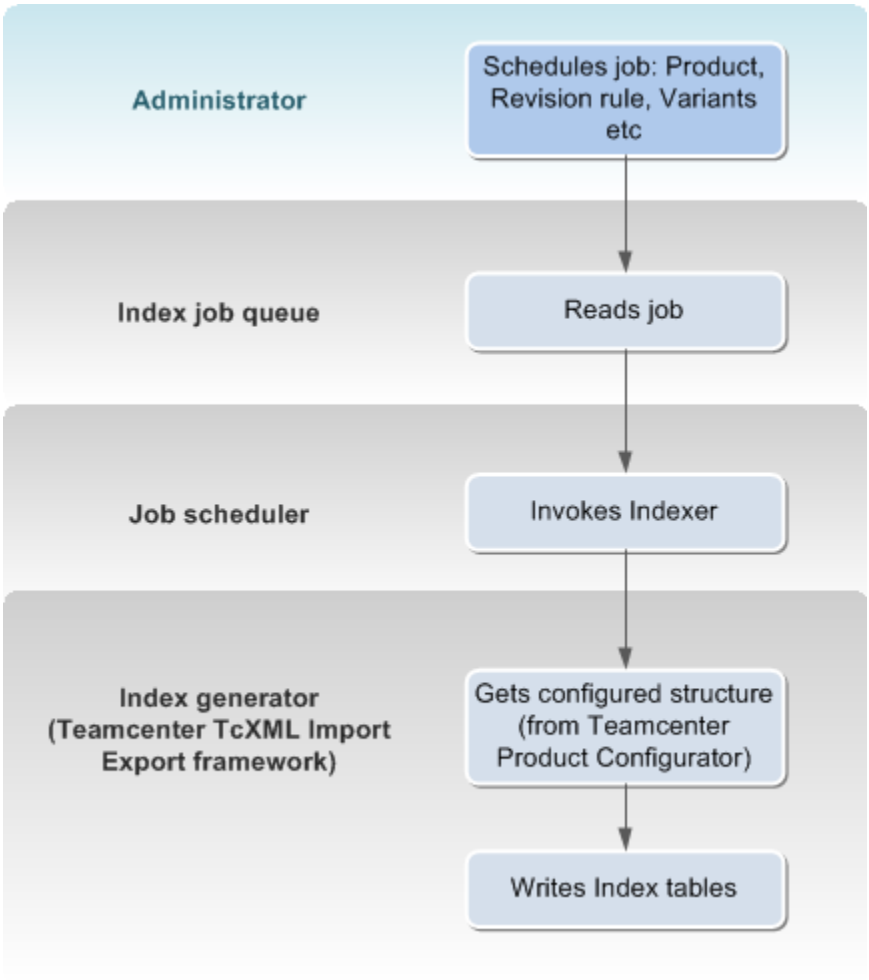
Index design and structured content

Teamcenter maintains a separate index for structured content to ensure search results are always up-to-date. The administrator defines the structures and configurations to index, minimizing response times whenever a user searches for data.

The following diagram shows the indexing framework mechanism.



The following diagram shows the process that occurs when you create an index for structured content.



To define structured content search indexes, you define property constants on structure elements in the Business Modeler IDE to specify the properties to index. The Teamcenter indexer then indexes those properties when it creates or updates the structured content index. You use the same property constants as you use with the basic search in Active Workspace but attach them to structure elements. Changing the structure filters follows the same process as other search filters.

You must reindex the structure at regular intervals to ensure that the search results reflect changes to the structure. Manual edits and system operations (for example, release and revise) on structures are reflected only after the indexer processes the changes.

Each top level of a structure is indexed for all the specified revision rules and effectivities. For example:

- **Latest Working, Released** creates one set of occurrences.
- **Released** creates one set of occurrences.
- **Released, Effectivity Milestone 1(1/1/2013)** creates one set of occurrences.

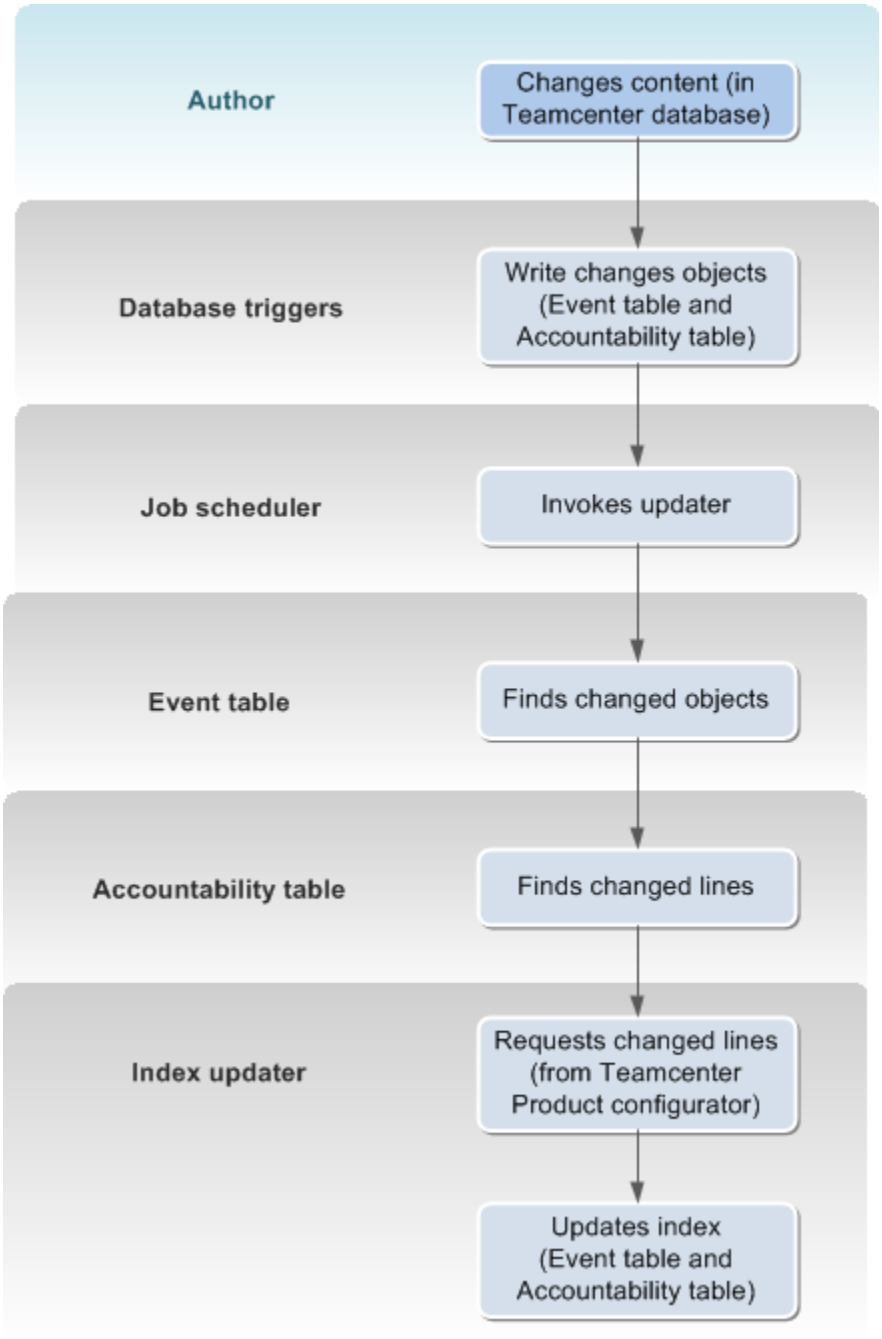
For best performance, Siemens Digital Industries Software recommends you index only those structures and configurations that are frequently accessed.

> Note:
>
> Whenever there is a change to a revision rule or a saved variant rule (SVR), you must **run an index synchronization on the structure**.

For each searchable revision rule, the system creates a unique set of occurrences that includes all possible variations in the structure (sometimes referred to as a 150% BOM). Each indexed occurrence is stored with a bit mask that identifies the valid variant rule for the occurrence. The variant mask is exported in TC XML format and indexed in the Teamcenter indexer. Teamcenter can then process the bit mask to build a 100% BOM for any given variant rule.

> Tip:
>
> To have the index update the variant mask when new SVRs are added to the index definition, set the **PersistFullyExpandedSVR** site preference to true before creating the SVRs.

The following diagram shows the process that occurs when a user changes index content and Teamcenter updates the index.



Teamcenter maintains the **indexed status of each structure** in the **Awb0BOMIndexAdminData** business object on the **awb0IndexState** property.

When do you need structure indexing?

Structure indexing is a prerequisite for the configuration and use of Massive Model Visualization (MMV) capabilities. Search filtering also requires structure indexing.

For additional use cases, the use of structure indexing may be considered after weighing the infrastructure and implementation costs in relation to the end user benefits. Consider the following examples of incremental benefits of structure indexing:

- The where-used search works without structure indexing. However, structure indexing additionally provides where-used capabilities in top-level contexts that are indexed.
- The keyword search is based on the object index and is cacheless. Structure indexing additionally provides facet count-based search refinement.

Your site administrators can help assess whether there is sufficient incremental value in additionally enabled search features or if the infrastructure and administration costs outweigh the benefits. If cacheless search with object index is sufficient for their mainstream use cases, they can make an informed decision to not use structure indexing although it is available.

Index structure content data

Index the structure content by running the **bomindex_admin** utility, for example:

```
bomindex_admin -u=username -p=password -g=dba -logfile=C:\Scratch\log\log1.txt
-function=create -inputfile=C:\Scratch\log\bomindex_admin_input.txt
```

Maintain the structure indexes by running the **runTcFTSIndexer** utility using the **structure** type.

Structure index life cycle

Over the course of an index life cycle there are 16 possible states. These states represent the current processing or condition the index is in.

Following is the life cycle of an index:

1. Created

   Product configuration parameters are defined but the index data is not yet generated.

2. Active

   The index data is generated and synchronized periodically.

3. Delete pending

   The product configuration is marked for deletion, or the delete processing is occurring.

4. Deleted

   There are no longer any artifacts relating to the index configuration.

The current index state and configuration details for product configurations are maintained within the Teamcenter database as unique **BOMIndexAdminData** table entries. As indexes for product configurations are created, maintained, and then eventually deleted, it is the **BOMIndexAdminData** table entry that tracks the configuration and state information. The data contained in the **BOMIndexAdminData** table is often referred to as product configuration, BIAD, or BIADInfo.

The **BOMIndexAdminData** entry for a given product configuration is created (and deleted) using the **bomindex_admin** command line utility.

The TcFTSIndexer is responsible for orchestrating the index generation and synchronization of the indexes defined by the **BOMIndexAdminData** entries.

Update an indexed structure with an added or modified saved variant rule (SVR)

Structure indexing supports adding new as well as modified saved variant rules on an existing indexed structure without a need to completely re-index the structure. When SVRs are saved, they do not automatically capture the default and derived default values. These values must be saved for each SVR that is used for indexing.

Log on to the Teamcenter client and set the **PersistFullyExpandedSVR** preference to **true**. Load the existing SVR in Structure Manager and click the **Save** button.

Index a structure with a closure rule

Administrators can index a structure with a closure rule to skip a subassembly or leaf BOM lines from structure indexing. Such lines do not appear in Active Workspace. These subassembly or leaf BOM lines do not appear in in-context search results and are not considered for index synchronization. The closure rule based filter can be applied per indexed configuration. The input file line format is as follows:

item-query-string | *item-revision-ID* | *base-revision-rule* | *effectivity-unit* | *effectivity-end-item-query-string* | *effectivity-date (dd-mmm-yyyy hh:mm:ss)* | *variant-rules* | *subscribers* | *closure-rules*

For example, if an administrator is applying a **BOMExpandSkipByItemType** closure rule while indexing with variant rules **vrule1** and **vrule2**, then the input file for the **bomindex_admin** utility is as follows:

```
item_id=HDD-0527 | B | Any Status; Working | 5 |
item_id=HDD-0527 | 31-May-2013 00:00:00 |
vrule1:item_id=OwnItem1:B,vrule2:,vrule3:item_id=OwnItem3:A | | BOMExpandSkipByItemType
```

If the closure rule is updated, removed, or replaced, the administrator must trigger re-index.

Overview of the bomindex_admin utility

The **bomindex_admin** utility creates the **BOMIndexAdminData** to define a specific product configuration to be indexed. This utility is also used to mark a specified product configuration (**/BOMIndexAdminData/BIAD**) for deletion.

A product configuration has many parameters, and they are passed to the utility in a text file specified by the **-inputfile** parameter. The input file is described in the **bomindex_admin** utility description.

You can specify up to 110 effectivities. The effectivity numbers must be comma separated. Also, you must repeat the effectivity end item query string for each effectivity unit, for example:

```
| 5,10,12 | item_id=HDD-0527,item_id=HDD-0527,item_id=HDD-0527 |
```

When performing the same function (action) on multiple product configurations, each product configuration can be specified on separate lines in the input file. Add these one at a time to help manage errors that may occur during the creation process.

> Note:
>
> Only **Working ()** and **Override Folder ()** revisions rules are supported. Any rules apart from these, for example, **Working (Owning User = Current)** or **Override Folder (Newstuff)**, are not supported. Additionally, **Any** revision status is not supported, either in the revision rule or in a custom revision rule entry for the input file. Find information on creating revision rules in the Teamcenter online help.

Structure indexing using TcFTSIndexer

The TcFTSIndexer is used to orchestrate the required processing to create, synchronize, and eventually delete the indexes defined by the **BOMIndexAdminData** tables (BIADs). TcFTSIndexer can be run in a number of ways and is also used to perform object data indexing.

The TcFTSIndexer has an extensibility model that allows different indexing types to define the processing required using a sequence of steps organized in flows. Steps can be reused by multiple flows and different indexing types. This is different than the traditional QETL model where every action is plugged in as a part of query, extract, transform, or load.

Key concepts of the extensibility model are:

- Type

  Specifies the name of the indexing type (for example, **objdata**, **structure**).

- Action

Specifies a name that ties a command line option to start a flow (for example, **sync** starts the synchronization flow).

- Flow

  Specifies the flow to execute (for example, **reindexflow**, **syncflow**).

- Step

  Specifies a single step that is executed as part of a flow.

With these concepts in mind, structure indexing is accomplished with:

- The **structure** indexing type.
- The **test**, **show**, **sync**, and **recoverfailures** actions.
- The **testflow**, **showflow**, **syncflow**, and **recoverfailuresflow** flows.
- A number of steps that are the building blocks of the various flows

The general syntax for starting any action is using the **runTcFTSIndexer** utility is:

```
runTcFTSIndexer -task=type:action [additional-arguments]
```

The **-help** argument lists all available actions. Do not use any actions described as **Internal use only**.

```
runTcFTSIndexer -help
```

If you are using multiple types of indexing (for example, object indexing and structure indexing), the TcFTSIndexer process must first be started in service mode, for example:

runTcFTSIndexer -background

However, you can still run the **-task=structure:show** and **-task=structure:help** tasks without having to start it as a service first.

The main **-task** actions for structure indexing are:

- **-task=structure:test**

  Performs basic tests, such as for Teamcenter log on, FMS connectivity, verifying or downloading of transform files, Solr schema, and so on. This command cannot be run concurrently with other structure indexing actions.

- **-task=structure:show**

Shows a summary of all configured product configurations.

- **-task=structure:sync**

  Performs normal synchronization and delete actions for all product configurations. This command queues up all the synchronization actions for the product configurations. The queued synchronization actions are processed as resource permits. This command cannot be run concurrently with other structure indexing actions. The TcFTSIndexer must be running in service mode to run this command.

- **-task=structure:syncone** *product-config-UID*

  Performs normal synchronization and delete actions for a single product configuration UID. This command queues up all the synchronization actions for the product configurations. The queued synchronization actions are processed as resource permits. This command cannot be run concurrently with other structure indexing actions.

- **-task=structure:recoverfailures**

  Changes all product configurations with failed states to the last known good state. Structure indexing will resume from the recovery state and structure will not be re-indexed completely.

  For example:

  IndexGenFailure will be changed to **ReadyToIndex**

  IndexExportFailure or **SolrIndexGenFailure** will be changed to **IndexExportRequested**

  IndexSyncExportFailure or **IndexGenSyncFailure** will be changed to **IndexSyncRequested**

  IndexDelFailure or **SolrIndexDelFailure** will be changed to **MarkedForDeletion**

- **-task=structure:resetall**

  Downloads the latest transform and schema files, resets all active product configurations to the **ReadyToIndex** state, and resets all deleted product configurations to the **MarkedForDeletion** state. This command cannot be run concurrently with other structure indexing actions.

- **-task=structure:reset** *product-config-UID*

  Resets the given **PRODUCT_CONFIG_UID** setting to the **ReadyToIndex** or **MarkedForDeletion** state. This command cannot be run concurrently with other actions.

Obtain TcFTSIndexer troubleshooting logs

TcFTSIndexer logs are located in *TC_ROOT***TcFTSIndexer\logs**. These logs contain messages from object and structure data as well as framework messages. For example:

- **TcFtsIndexer_objdata.log** contains object data messages. These messages are filtered from **TcFtsIndexer.log**.
- **TcFtsIndexer_structure.log** contains structure data messages. These messages are filtered from **TcFtsIndexer.log**.

To obtain the troubleshooting logs:

1. Change the logging level to **DEBUG** in the following **%TC_DATA%\logger.properties** file:
   - **logging.rootLogger**
   - **logging.logger.Teamcenter**
   - **logging.logger.Teamcenter.Soa.Communication**
2. Change the logging level to **DEBUG** for the following in the **%TC_ROOT%\TcFTSIndexer\conf\log4j.properties** file:
   - **Log4j.logger.com.siemens.teamcenter.ftsi**
3. Restart Solr and the **tcserver** instances and rerun the test use case.
4. Use the **TcFtsIndexer log** and the matching **syslog** and **comlog** files for troubleshooting or reporting.

   You can identify the **syslog** and **comlog** files by matching the local time in the **TcFtsindexer** log where the error occurred with the UTC time within the syslogs. Send all files that fit this criteria.

> Note:
>
> To avoid performance issues, revert the logging levels back to the original state when your debug session is complete.

Resolve TcFTSIndexer issues

Issue	Possible resolution
Locate errors in **TcFTSIndexer**	**TcFTSIndexer** logs are located in *TC_ROOT***TcFTSIndexer\logs**. Choose a method for finding errors that most closely aligns with your issue. • If the **TcFTSIndexer** is still running, you can send a summary log report to the command window and to the **TcFtsIndexer.log**. Open a new shell and run:

Issue	Possible resolution
	• **TcFtsIndexer.bat/sh -status** to generate the summary report in the console. The summary shows the steps in the flow where errors occurred. • **TcFtsIndexer.bat/sh -debug** to generate additional information in the summary report. The summary shows the flow in progress, including connections and the logs associated with them. • If the **TcFTSIndexer** finished processing, navigate to the end of **TcFtsIndexer.log**. The report contains an entry for each **TaskId** that has an error. Search for the **TaskId** in the log to locate the point of failure to learn more about the error. Note: If you need more information or if a file is referenced in the error, you can search for the **TaskId** in the files associated with the error, including the files from the previous step, in the **TcFtsIndexer\working** directory. For example, if the error is in **Transform**, the associated directory contains export files and transform files that you can use to resolve the error.
Indexing performance	Indexing performance depends on the number of warmed-up **tcserver** instances and the number of connections to those servers that are available for indexing. Using more servers and connections supports greater parallelization. Note: To ensure the optimal number of warmed up servers, Siemens Digital Industries Software recommends that the pool manager that maintains the **tcservers** be setup on a separate, dedicated machine. You can edit the **Tc.maxConnections** property in the **TcFtsindexer\conf\TcFtsIndexer.properties** file to specify the maximum number of **Tc** connections open simultaneously. You can also change this value dynamically: 1. Open a new Teamcenter command window and navigate to the *TC_ROOT***TcFTSIndexer\bin** directory.

Issue	Possible resolution
	2. Run the following **runTcFTSIndexer** utility command, where the value for *connections* is the number of connections desired: `runTcFTSIndexer -maxConnections=`*connections* The *connections* value should never exceed the number of warmed up servers. Note: If you receive **WARN - Connection to Tc failed** messages, check to ensure the number of **Tc.maxConnections** has not exceeded the number of warmed servers.
Login error	You may encounter the following error when attempting to run the **runTcFTSIndexer** utility and the environment is configured for SSO: `Login Error: The login attempt failed: either the user ID or the password is invalid.` It may occur because the user running the utility is not properly authenticated in the LDAP server. The default user that runs the utility is a user with Teamcenter administrative privileges, as defined in the **Tc.user** setting in the *TC_ROOT***TcFTSIndexer\conf\TcFtsIndexer.properties** file. Ensure that the user running the indexer is authorized in LDAP: 1. If you are using multiple TCCS SSO App IDs, make sure they are configured correctly. You can configure multiple application IDs using the **Environment Settings for Client Communication System** panel in Teamcenter Environment Manager (TEM). 2. Ensure that the user defined by the **Tc.user** setting in the *TC_ROOT***TcFTSIndexer\conf\TcFtsIndexer.properties** file is a valid user in the LDAP server and the Teamcenter database. Create a user in both if needed, or select an existing valid active user to run the **runTcFTSIndexer** utility.

Issue	Possible resolution
	3. In the console, set an environment variable to the password value. `set` *mytcenv=password* 4. Create an encrypted password file for this user by running the **encryptPass.bat/sh** utility, located in the *TC_ROOT***TcFTSIndexer\bin** directory, with the **-tc** argument and specifying the environment variable name created in the previous step, for example: `encryptPass -tc` *mytcenv* 5. After you create the encrypted password file, remove the environment variable value. `set` *mytcenv=*
TcFTSIndexer output states the search engine is not accessible	The following message is displayed in output after running the **runTcFTSIndexer** utility: `ERROR - The search engine is not accessible or the search engine schema is not correct.` The Solr schema needs to be updated. Use the following command: *SOLR_HOME*`\TcSchemaToSolrSchemaTransform.bat` *TC_DATA*`\ftsi\solr_schema_files`
tcservers run out of memory	Reduce the **Tc.maxConnectionUsedCount** value in the **TcFtsindexer\conf\TcFtsIndexer.properties** file to reduce the number of times a **tcserver** connection can be reused before log out. This helps to lower the memory consumption per **tcserver**.
tcserver Solr authentication error	Error: **An error has occurred during JSON parsing: Unknown value type. Line 1 character 1.** To resolve this error, update Solr credentials to reset the Solr password.

Troubleshoot structures

Issue	Possible resolution

TcFTSIndexer output indicates a type definition does not exist	The following message is displayed in output after running the **runTcFTSIndexer** utility:
	
	`Type definition for type does not exist.`
	`Supported Type [objdata]`
	
	The **Active Content Structure** features are not installed. Rerun the Teamcenter Environment Manager(TEM) and select all **Active Content Structure** features.

Indexes are in failure states.	Use the **-task=structure:recoverfailures** argument. Examine the log output for details on the failures.
Changes to structured content do not appear immediately in searches.	Changed data is shown immediately in searches when users add, remove, or change elements (occurrences or BOM lines). However, when users change the underlying objects to which occurrences or BOM lines refer, the changed data is not shown immediately in the content. This includes: • Revisions to the underlying object. • Releasing the underlying object. • Changing effectivity on the release status. • Changing properties on the underlying object. Users see these changes in the content the next time data is indexed. Note: The interval between indexing synchronizations is set by the search administrator.
Structure indexing fails for **Working (Owning User = Current)** revision rule.	Only **Working ()** and **Override Folder ()** revisions rules are supported. Any rules apart from these, for example, **Working (Owning User = Current)** or **Override Folder (Newstuff)**, are

Issue	Possible resolution
	not supported. Use a revision rule other than these for the structure indexing to work.

Overview of managing structure indexes

To manage a given structure index, it first must be created using the **bomindex_admin** utility with the **-function=create** option.

Once the product configuration is created, the TcFTSIndexer service automatically maintains the indexes keeping them synchronized with the Teamcenter data that defines the structure. The index is synchronized periodically based on the interval the synchronization operation runs using the following **runTcFTSIndexer** utility command:

```
runTcFTSIndexer -task=structure:sync -interval=seconds
```

Once the product configuration is no longer needed, the **bomindex_admin** utility is used to mark the product configuration for deletion using the **-function=delete** option. On the next synchronization interval, the TcFTSIndexer process deletes the indexes and then finally deletes the associated **BOMIndexAdminData** object that defined the product configuration.

Create a structure index

To create an index for a specific product configuration, use the **bomindex_admin** utility with the **-function=create** and **-inputfile** options. Determine the product configuration that defines the index. Create an input file with the format specified, configuring the index.

The input file line format is as follows:

item-query-string | *item-revision-ID* | *base-revision-rule* | *effectivity-units* | *effectivity-end-item-query-strings* | *effectivity-dates (dd-mmm-yyyy hh:mm:ss)* | *variant-rules* | *subscribers* | *closure-rules*

You can specify up to 110 effectivities. The effectivity numbers must be comma separated. Additionally, you must repeat the effectivity end item query string for each effectivity unit, for example:

```
| 5,10,12 | item_id=HDD-0527,item_id=HDD-0527,item_id=HDD-0527 |
```

You can specify up to 256 variant rules. The variant rules (also known as saved variant rules) are comma separated and follow this format:

SVR-name:owning-item-query-string:owning-itemrevision-ID

The top line item revision is the default owner.

An example of an input file (**bomindex_admin_input.txt**) is as follows:

```
item_id=HDD-0527 | B | Any Status; Working | 5 |item_id=HDD-0527 | 31-May-2013 00:00:00
|
vrule1:item_id=OwnItem1:B,vrule2:,vrule3:item_id=OwnItem3:A
```

An example of running the **bomindex_admin** utility is as follows:

```
bomindex_admin -u=username -p=password -g=dba -function=create
-inputfile=bomindex_admin_input.txt -logfile=bomindex_admin.log
```

Assuming that there are no errors, this creates the required **BOMIndexAdminData** entries for the specified product configuration. At this point, the configuration required to generate and maintain the indexes exists, but there is no actual index data.

Save the input file for later use when the product configuration must be deleted.

Delete a structure index

When a given product configuration is no longer needed, mark the index for deletion so that the index artifacts can be cleaned up. Use the **bomindex_admin** utility to mark the product configuration for deletion using the same input file contents when the index was created.

Following is an example input file named **bomindex_admin_input.txt**:

```
item_id=HDD-0527 | B | Any Status; Working | 5 |item_id=HDD-0527 | 31-May-2013 00:00:00
|
vrule1:item_id=OwnItem1:B,vrule2:,vrule3:item_id=OwnItem3:A
```

Following is the example **bomindex_admin** utility execution:

```
bomindex_admin -u=username -p=password -g=dba -function=delete
-inputfile=bomindex_admin_input.txt -logfile=bomindex_admin.log
```

Assuming there are no errors, this marks the **BOMIndexAdminData** entries for deletion. On the next **sync** action, the index data and **BOMIndexAdminData** table entry are deleted. At the end of the **sync** action (delete), the **show** output is printed to the console:

```
--- BOM Index Summary ---
Status Product Config UID Window UID     State Name                          TC
Count     Solr Count
------ ------------------ -------------- ----- ------------------------------
------------ ------------
AC     gcRNr4APIWcIeC     9Axq$7ymM1y6BD     8 SUB-479/A;1-holland-assy
2,254         2,254
AC     gcUNanthIWcIeC     $JCJg8_$M1Cq9C     8 flipfone_assembly/A;1
14            14
```

```
------ ----------------- ------------- ----- ----------------------------
------------ ------------
```

View the current states of the structure indexes

Little should be required to maintain the structure indexes. The TcFTSIndexer service handles all processing as part of the **-task=structure:sync -interval=***seconds* processing.

It is good practice to verify index states occasionally by using the **-task=structure:show** action. The **show** action prints a summary of information to the service console of all product configurations configured for indexing. Details of the configured indexers is shown in output as well as counts and status. When synchronization is not actively processing an index, all states should be **8**. (See the following example.) If synchronization is actively processing a given index, the index can also be in various intermediate states. Within the **show** output, verify the state, the last update date, and that the TC counts and Solr counts match.

Run the following **runTcFTSIndexer** utility command:

```
runTcFTSIndexer -task=structure:show
```

The following example output in the service console shows all indexes are in state 8. The counts match and the last update dates are current.

```
2014-08-07 13:17:45,442 INFO  - Running TcFtsIndexer Type: structure FlowAction: show
--- BOM Index Summary ---
Status Product Config UID Window UID     State Name                          TC
Count     Solr Count
------ ----------------- ------------- ----- ----------------------------
------------ ------------
AC     wkUN927DoR4_1D     StimEfzjM1SdMD     8 HDD-0527/A;1-Hard Drive Assemb
364          364
AC     wkXN927DoR4_1D     bCCyVUKhM1SoNA     8 HDD-0527/A;1-Hard Drive Assemb
364          364
AC     wkaN927DoR4_1D     oMOQnKj5M1yEfC     8 HDD-0527/B;1-Hard Drive Assemb
195          195
AC     wkdN927DoR4_1D     lwfEyXsKM1i0TB     8 HDD-0527/B;1-Hard Drive Assemb
195          195
AC     wseN927DoR4_1D     SDFtNH5NM1y6wC     8 JCB-Fastrac/B;1-Tractor
9,968        9,968
AC     wwRN927DoR4_1D     3GOvLJUYM1yOyB     8 JCB-Fastrac/B;1-Tractor
9,968        9,968
------ ----------------- ------------- ----- ----------------------------
------------ ------------
```

> Note:
>
> The **show** action does require available Teamcenter connections. If the **show** command takes a long time to run it usually is the result of other actions within the indexer using the connections. Once a connection becomes available, the **show** action runs.

The **show** output currently is only displayed in the TcFTSIndexer service console. The **show** output is also printed at the end of many of the structure indexer actions.

View the status of synchronization in progress

Another useful **runTcFTSIndexer** option is **-status**. This prints status information about any currently running flows. The output in the **syncdispatchstep** sections provides details about current **structure:sync** processing.

Run the following command:

```
runTcFTSIndexer –status
```

The following output was gathered while the **structure:sync** command was in process. For **structure:sync** actions, the most useful information is contained in the **syncdispatchstep** output sections. Run the **runTcFTSIndexer -status** command to see output similar to the following example:

```
----------------------------------------
syncdispatchstep----------------------------------------
          TaskId             Time      Status          StepInfo
U14977b73edd1c0a802641379    0.00    Started
{ProductConfigUID=goZRkWxoqd$DyB, WindowUID=nmSCFWD0M1SBXC,
Process=sync (update),
Status=SOA call in progress, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:42 -0500}
----------------------------------------------------------------------------------------
-----------
Status: Created: 0   Started: 1   Done: 0   Error: 0
Total Time    0.00   Total Count 0
Step Summary
  syncdispatchstep
     Status: Created: 0   Started: 1   Done: 0   Error: 0
Total time for all Steps 0 sec
Overall Time 5.314 sec
----------------------------------------
syncdispatchstep----------------------------------------
          TaskId             Time      Status          StepInfo
U1493c8428cd7c0a802641381    0.00    Started  {ProductConfigUID=gocRkWxoqd$DyB,
WindowUID=ybqRdpVmM1Sn1A,
Process=sync (update),
Status=Waiting for 1 permits, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:39 -0500}
----------------------------------------------------------------------------------------
-----------
Status: Created: 0   Started: 1   Done: 0   Error: 0
Total Time    0.00   Total Count 0
Step Summary
  syncdispatchstep
     Status: Created: 0   Started: 1   Done: 0   Error: 0
Total time for all Steps 0 sec
Overall Time 5.469 sec
...
```

Because the sections and status output is scattered you may want to filter the output to show only the status lines. Run the **runTcFTSIndexer -status | find "ProductConf"** command to see output similar to the following example:

```
U14977b73edd1c0a802641379    0.00    Started  {ProductConfigUID=goZRkWxoqd$DyB,
WindowUID=nmSCFWD0M1SBXC, Process=sync (update),
Status=SOA call in progress, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8, 5,
8],
LastStatusUpdate=2014-10-27 17:02:54 -0500}

U1493c8428cd7c0a802641381    0.00    Started  {ProductConfigUID=gocRkWxoqd$DyB,
WindowUID=ybqRdpVmM1Sn1A, Process=sync (update),
Status=SOA call in progress, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:54 -0500}

U14949b7e5e27c0a802641383    0.00    Started  {ProductConfigUID=gofRkWxoqd$DyB,
WindowUID=OVSe3Sn0M1CE$B, Process=sync (update),
Status=Waiting for 1 permits, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:39 -0500}

U1499a46af6c7c0a802641387    0.00    Started  {ProductConfigUID=gsSRkWxoqd$DyB,
WindowUID=4weJYgJ9M1yOVC, Process=sync (update),
Status=Waiting for 1 permits, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:39 -0500}
...
```

This output shows the various product configurations, what initial state they were in, the states they were propagated through (for this **sync** action), and other status information about what types of operation are currently pending such as waiting on connection, SOA call in progress, or done.

Manage failures

When a product configuration ends up in a failed state, it remains in that state until the administrator runs the **structure:recoverfailures** action. When that is run, any failed product configurations are returned to the initial state and the index is regenerated (re-indexed) on the next synchronization.

Run the following command:

```
runTcFTSIndexer -task=structure:recoverfailures
```

Following is truncated example output. It shows a failed product is now in state **0**.

```
--- BOM Index Summary ---
Status Product Config UID Window UID     State Name                              TC
Count     Solr Count
------ ------------------ -------------- ----- ------------------------------
------------ ------------
AI     gcRNanawIWcIeC     s3PvF$$yM1ipFC     0 HDD-0527/B;1-Hard Drive Assemb
81           81
AC     gcRNr4APIWcIeC     9Axq$7ymM1y6BD     8 SUB-479/A;1-holland-assy
2,254        2,254
AC     gcUNanthIWcIeC     $JCJg8_$M1Cq9C     8 flipfone_assembly/A;1
14           14
```

```
------ ----------------- ------------- ----- ----------------------------
------------ ------------
```

On the next synchronization interval, that product configuration's indexes regenerated, and in this case, it succeeds. Following is the truncated output:

```
--- BOM Index Summary ---
Status Product Config UID Window UID     State Name                          TC
Count     Solr Count
------ ----------------- ------------- ----- ----------------------------
------------ ------------
AC     gcRNanawIWcIeC     s3PvF$$yM1ipFC     8 HDD-0527/B;1-Hard Drive Assemb
81           81
AC     gcRNr4APIWcIeC     9Axq$7ymM1y6BD     8 SUB-479/A;1-holland-assy
2,254        2,254
AC     gcUNanthIWcIeC     $JCJg8_$M1Cq9C     8 flipfone_assembly/A;1
14           14
------ ----------------- ------------- ----- ----------------------------
------------ ------------
```

If a product configuration continues to fail, output generated during the synchronization processing, TcFTSIndexer logs, and **tcserver syslog** files should help diagnose the underlying issue.

A common source of errors is stopping the TcFTSIndexer while synchronization operations are in progress. If you wish to stop the TcFTSIndexer *you should never kill the process while actions are being processed*. Use the **-stop** option to stop the scheduling of any flows, then verify that all flows have stopped using the **-status** option, and then finally shut down the TcFTSIndexer process using Ctrl+C in the service console window.

Restart the indexer

Sometimes you may observe that the FTS indexer is stuck. The symptoms of this condition are one or more of the following:

- The indexer is not picking a product for indexing.
- When you run **TcFTSIndexer.bat** with the **-status** argument, it reports status as `Status=SOA call in progress`. This status does not change over a long period of time.
- One or more Teamcenter server process crashes is observed in the logs.

These symptoms indicate the Teamcenter server crashed while processing and the crash was not reported back to the FTS indexer by the pool manager or Web tier.

To resolve this condition, you should stop synchronization, then (after an interval) restart the **TcFTSIndexer** service and reset the stuck configuration.

Structure index states

There are various categories of index states:

- Initial

  Indicates either new index creation or that an existing index is marked for deletion. In the state output, the create state is **0 - ReadForIndexing** and the marked for deletion state is **10 - MarkedForDeletion**. These states are usually set by the **bomindex_admin** utility, but can also be set by certain TcFTSIndexer actions. When the next **sync** action occurs, any product configuration in these states is propagated to other states during processing.

- Transitional

  Tracks the intermediate progress while processing a given product configuration. Transitional states should naturally propagate to final or terminal (failed) states during **sync** processing. Transitional states are only expected to be encountered during **sync** processing. If a transitional state is encountered at the start of **sync** processing, that indicates the given product configuration was not able to complete index processing for an unknown reason and that product configuration is promoted to the closest failure state.

- Final

  Indicates index processing for the given product configuration completed normally (**8 - SolrIndexGenSuccess**). Although you might expect to have a final deleted success state as well, the final step in index delete processing is to delete the product configuration data for the index, along with the state information.

- Terminal

  Track failures at particular steps in index processing. Once these states are reached, no further processing occurs for the given product configuration. Failure states are set either:

  - Directly as a result of a **processBOMIndex** SOA call when the **tcserver** process had an issue processing the index. Examine the **tcserver** syslogs for details about the failure. Examine the **TcFtsIndexer.log** file for the **ERROR** message and any other information that was sent back with the error.

  - Or the failure was detected in the TcFTSIndexer process due to a failure or an unexpected event. Examine the **TcFtsIndexer.log** file or the **TcFtsIndexer_structure.log** file for **ERROR** information.

  To recover from terminal states (failures) use the **structure:recoverfailures** action, which resets all failed product configurations to the appropriate initial state and the next **sync** action attempts to process them again.

You must be careful that TcFTSIndexer is never stopped while the **structure:sync** action is running. If it is stopped prematurely any indexes that were still being processed are in transitional states and are set to terminal states on the next **sync** action.

Complete structure index state list

Run the following **runTcFTSIndexer** utility command to see the list of structure index states:

```
runTcFTSIndexer -task=structure:show
```

Following is the complete list of index states:

0 – ReadyToIndex

Indicates an initial index state usually set by the **bomindex_admin -function=create** command. This state is also set by the **structure:reset**, **structure:resetall**, or **structure:recoverfailures** actions for active but failed indexes.

1 – IndexGenStarted

Indicates a transitional state set by the **processBOMIndex** SOA operation. It is set while the Teamcenter database index generation is in progress.

2 – IndexGenSuccess

Indicates a transitional state set by the **processBOMIndex** SOA operation. It is set when the Teamcenter database index generation is complete.

3 – IndexGenFailure

Indicates a terminal failure state set by the **processBOMIndex** SOA operation. It is set if the Teamcenter database index generation failed.

4 – IndexExportStarted

Indicates a transitional state set by the **processBOMIndex** SOA operation. It is set while the Teamcenter database index export is in progress.

5 – IndexExportSuccess

Indicates a transitional state set by the **processBOMIndex** SOA operation. It is set while the Teamcenter database index export is complete and the TC XML download is starting.

6 – IndexExportFailure

Indicates a terminal failure state set by the **processBOMIndex** SOA operation. It is set if the Teamcenter database index export or TC XML download fails.

7 – SolrIndexGenStarted

Indicates a transitional state set by structure indexer. It is set while the TcFTSIndexer is transforming and uploading the index data to Solr.

8 – SolrIndexGenSuccess

Indicates a final resting state set by structure indexer. It is set when the TcFTSIndexer has successfully updated Solr with the index data.

9 – SolrIndexGenFailure

Indicates a terminal failure state set by structure indexer. It is set if the TcFTSIndexer had a failure while transforming or uploading the index data.

10 – MarkedForDeletion

Indicates an initial delete state usually set by the **bomindex_admin -function=delete** command. It is also set by the **structure:recoverfailures** action for any deleted indexes that failed during delete processing.

11 – IndexDelStarted

Indicates a transitional state set by the **processBOMIndex** SOA operation. It is set while the Teamcenter database index data is being deleted.

12 – IndexDelSuccess

Indicates a transitional state set by the **processBOMIndex** SOA operation. It is set when the Teamcenter database index data delete is complete.

13 – IndexDelFailure

Indicates a terminal failure state set by the **processBOMIndex** SOA operation. It is set if the Teamcenter database index data delete failed.

14 – SolrIndexDelStarted

Indicates a transitional state set by the structure indexer. It is set while the TcFTSIndexer is deleting the Solr index data.

15 – SolrIndexDelSuccess

Indicates a transitional state set by the structure indexer prior to object deletion. After this state, the **BOMIndexAdminData** entry is deleted.

16 – SolrIndexDelFailure

Indicates a terminal failure state set by the structure indexer. It is set if the TcFTSIndexer had a failure while deleting the Solr index data.

17 – IndexGenSyncStarted

Indicates a transitional state set by the **processBOMIndex** SOA operation when an index was previously synchronized and is currently being synchronized again. (This is a synchronization update as opposed to an initial synchronization.)

Structure index state propagation

This is the actual state propagation that occurs when processing indexes. Not all of these states are necessarily seen in the indexer as many are expected to be transitioned during backend server processing. Following is a complete set of state propagations for various types of **sync** processing:

- **sync** (initial)

  0 - ReadyToIndex
  1 – IndexGenStarted (On failure, processing goes to **3 – IndexGenFailure** and then stops.)
  2 – IndexGenSuccess
  4 – IndexExportStarted (On failure, processing goes to **6 – IndexExportFailure** and then stops.)
  5 – IndexExportSuccess
  7 – SolrIndexGenStarted (On failure, processing goes to **9 – SolrIndexGenFailure** and then stops.)
  8 – SolrIndexGenSuccess (Processing then stops.)

- **sync** (update)

  8 – SolrIndexGenSuccess
  17 - IndexGenSyncStarted (This is a transitional state set by the **processBOMIndex** SOA operation.)
  4 – IndexExportStarted (On failure, processing goes to **6 – IndexExportFailure** and then stops.)
  5 – IndexExportSuccess
  7 – SolrIndexGenStarted (On failure, processing goes to **9 – SolrIndexGenFailure** and then stops.)
  8 – SolrIndexGenSuccess (Processing then stops.)

- **sync** (delete)

  10 - MarkedForDeletion

11 – IndexDelStarted (On failure, processing goes to **13 – IndexDelFailure** and then stops.)
12 – IndexDelSuccess
14 – SolrIndexDelStarted (On failure, processing goes to **16 – SolrIndexDelFailure** and then stops.)
15 – SolrIndexDelSuccess (The entry is deleted.)

Show structure index output

Run the following **runTcFTSIndexer** utility command to see the structure index status:

```
runTcFTSIndexer -task=structure:show
```

Following is sample output:

```
--- BOM Index Summary ---
Status Product Config UID Window UID     State Name                          TC
Count     Solr Count
------ ------------------ -------------- ----- ------------------------------
------------ ------------
AC     wkUN927DoR4_1D     StimEfzjM1SdMD     8 HDD-0527/A;1-Hard Drive Assemb
364          364
AC     wkXN927DoR4_1D     bCCyVUKhM1SoNA     8 HDD-0527/A;1-Hard Drive Assemb
364          364
AC     wkaN927DoR4_1D     oMOQnKj5M1yEfC     8 HDD-0527/B;1-Hard Drive Assemb
195          195
AC     wkdN927DoR4_1D     lwfEyXsKM1i0TB     8 HDD-0527/B;1-Hard Drive Assemb
195          195
AC     wseN927DoR4_1D     SDFtNH5NM1y6wC     8 JCB-Fastrac/B;1-Tractor
9,968        9,968
AC     wwRN927DoR4_1D     3GOvLJUYM1yOyB     8 JCB-Fastrac/B;1-Tractor
9,968        9,968
------ ------------------ -------------- ----- ------------------------------
------------ ------------
```

Following is an explanation of the columns:

- **Status**

  Indicates the general category of the index state. The output is sorted by the status category.

  - Active index status codes:

    - **AI**

      Flagged for initial index generation or re-index.

    - **AP**

      Indicates that **sync** process in progress.

- **AC**

  Indicates that the **sync** process is complete.

- **AF**

  Indicates that the **sync** process failed.

- Deleted index status codes:

  - **DI**

    Flagged for deletion.

  - **DP**

    Indicates that deletion is in progress.

  - **DF**

    Indicates that the deletion failed.

- **Product Config UID**

  Indicates the UID of the **BomIndexAdminData** object that identifies the product configuration details.

- **Window UIDs**

  Indicates the BOM window UID.

- **State**

  Indicates the state of the index. A state legend is included in the show output.

- **Name**

  Indicates the name of the top line of the product structure.

- **TC Count**

  Indicates the number of occurrences found in the database for this product configuration.

- **Solr Count**

  Indicates the number of occurrences found in Solr for this product configuration. (The counts should match.)

- **Last update date**

  Indicates the time when the index was last updated.

Status syncdispatch output

Run the following command:

```
runTcFTSIndexer -status | find "ProductConf"
```

Following is an example of the output in the **syncdispatchstep** section filtered to show only the structure status information:

```
U14977b73edd1c0a802641379    0.00    Started  {ProductConfigUID=goZRkWxoqd$DyB,
WindowUID=nmSCFWD0M1SBXC, Process=sync (update),
Status=SOA call in progress, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8, 5,
8],
LastStatusUpdate=2014-10-27 17:02:54 -0500}

U1493c8428cd7c0a802641381    0.00    Started  {ProductConfigUID=gocRkWxoqd$DyB,
WindowUID=ybqRdpVmM1Sn1A, Process=sync (update),
Status=SOA call in progress, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:54 -0500}

U14949b7e5e27c0a802641383    0.00    Started  {ProductConfigUID=gofRkWxoqd$DyB,
WindowUID=OVSe3Sn0M1CE$B, Process=sync (update),
Status=Waiting for 1 permits, CurrentState=8 (SolrIndexGenSuccess), StateHistory=[8],
LastStatusUpdate=2014-10-27 17:02:39 -0500}
...
```

Following is an explanation of the columns:

- **ProductConfigUID**

  Indicates the UID of the **BomIndexAdminData** object that identifies the product configuration details.

- **WindowUID**

  Indicates the BOM window UID.

- **Process**

  Indicates the type of processing that is occurring for the given product configuration. Following are available values:

  - **Dispatching**

    Indicates that dispatching is occurring. This is a transient message.

- **sync (initial)**

  Indicates the first time the index is generated (for example, during re-index).

- **sync (update)**

  Indicates an incremental **sync** update is occurring.

- **sync (delete)**

  Indicates **sync** delete processing.

- **promoting to failure (found in intermediate state)**

  Indicates that the given product configuration did not complete processing on the last **sync** action and is being promoted to a failure state.

- **ignored (previously failed)**

  Indicates that the given product configuration was previously promoted to a failure state and is being skipped during this processing.

- **Status**

  Indicates fine-grained detail about the current processing. Possible values include:

  - **Waiting for # permits**

    Indicates that processing is waiting for the required number of permits to begin processing.

  - **Waiting for connection**

    Indicates that processing is waiting for a Teamcenter connection.

  - **SOA call in progress**

    Indicates that a Teamcenter SOA call is in process.

  - **Download**

    Indicates that export files are being downloaded.

  - **Transform**

    Indicates that export data is being transformed into Solr files.

- **Solr**

  Indicates that Solr is being updated.

- **Done**

  Indicates that process completed without error.

- **Failed**

  Indicates that an error occurred.

- **CurrentState**

  Indicates the current state of the given product configuration.

- **StateHistory**

  Indicates the history of the states recorded while processing this product configuration during this **sync** process. Some expected values are **[8, 5, 8]** (**sync** update) and **[0, 2, 5, 8]** (initial **sync**).

Important structure content indexing files

The following files are used in structure content indexing:

- Configuration file

  TC_ROOT\TcFTSIndexer\conf\TcFtsIndexer_structure.properties file

  This properties file configures the structure indexing including the type, flows, steps, and actions. Most of the content in this file should not be modified. There is one set of properties that configures control logic to limit the number of concurrent sync operations, weighting the different types of **sync** actions (initial versus update) differently. The reason for the different weighting is that initial **sync** processing is a much more resource intensive operation on the **tcserver** instance and database processes.

  Default values are shown in the file as follows:

  ```
  #
  # control the number of concurrent structure syncs to limit server and oracle load
  #
  permits.total=7
  permits.required.initialsync=3
  permits.required.updatesync=1
  ```

  For a given type of **sync** actions to run (initial or update), it must be able to obtain the required number of permits for that type. There is a limited (total) number of permits that can be used at

any given time. Any **sync** operation that cannot obtain the required permits waits until other **sync** processing finishes, releasing more permits. This controls the load on the servers during the **sync** processing.

The combinations of **sync** processes that can run concurrently:

0 **initialsync** actions and up to 7 **updatesync** actions

1 **initialsync** actions and up to 4 **updatesync** actions

2 **initialsync** actions and only 1 **updatesync** action

When making changes to these values also consider the maximum number of connections properties found in the **TcFtsIndexer.properties** file.

- Logging configuration file

  TC_ROOT\TcFTSIndexer\conf\log4j.properties

  This file is used to change the logging level. (The default logging level for the TcFTSIndexer process is **INFO**.)

- Log file

  TC_ROOT\TcFTSIndexer\logs\TcFtsIndexer.log

- Cache files

  TC_ROOT\TcFTSIndexer\cache*.cache

  These files may need to be deleted if Teamcenter preferences are changed that affect the Solr installation and FMS.

Integrate a new search indexing type with dispatcher

If there are steps that need to be offloaded to a different machine for CPU or memory optimizations, mark them as *step-name***.usedispatcher=true** in the property file. Each step must implement the dispatcher-specific interfaces. These methods are called by the TcFTSIndexer framework from the TcFTSIndexer orchestrator or the Dispatcher module depending on the context.

1. Make sure that the TcFTSIndexer is installed and working in dispatcher mode.

2. Create a dispatcher flow that calls the step to be run in the Dispatcher module in the **TcFtsIndexer_***type-name***.properties** file.

3. Copy the custom **TcFtsIndexer_***type-name***.properties** file to the *dispatcher-location***Module\Translators\TcFTSIndexer\conf** directory.

4. Copy the custom JAR file to the *dispatcher-location***Module\Translators\TcFTSIndexer\lib** directory.

5. Create a new translator service in the *dispatcher-location***Module\conf\translator.xml** file that executes the new dispatcher flow. See the existing example **objdatatransformstep** service for details.

6. Start the Dispatcher module.

7. Run the main flow from TcFTSIndexer orchestrator using the **-dispatcher** argument.

12. Configure structure search

About the different structure search and their required configurations

There are several way to search for elements within a structure in Teamcenter. Some of the searches require you to perform certain configurations and to index structure. As per your site requirements, you can choose which ones to configure for your users.

Search type	Supported on	Description	Supports	Does not support	Required configuration
Find components in display	Rich client	Searches for an element within a structure based on the search parameters entered in the **Find in Display** dialog box.	NA	• Spatial search • Classification search • Find using saved queries Additionally, the search cannot be saved, replayed, interoperated, or shared with other users.	Available by default. No specific configuration or structure indexing is required.
Cacheless search	Rich client	Searches for an element within a structure based on the search parameters entered in the **Search** dialog box.	• Spatial search • Classification search	NA	Requires a Context Management User license. Index structures by using Cacheless search. You can use one of the following methods: • Use the *Full database* method if the Teamcenter database is small in size. • Use the *Product-scoped cacheless search* method to index specific
	Active Workspace	Searches for structures based on the search parameters entered in the **Find** panel on performing			

Search type	Supported on	Description	Supports	Does not support	Required configuration
		a **Quick** search.			structures if the Teamcenter database is medium or large in size.
Keyword find and filter	Active Workspace	Searches for an element within a structure based on the search parameters entered in the **Find** panel.	For structures indexed using Active Content Structure indexing, supports: • Color filters • Massive Model Visualization	For structures indexed using Active Content Structure indexing, does not support: • Visualization of filtered structures • Spatial search • Classification search Additionally, the filtered structure cannot be saved, replayed, interoperated, or shared with other users.	Index structures by using **Active Content Structure indexing**. Warning: The Active Content Structure indexing feature is soon going to be deprecated. You must use this indexing only for massive model visualization. Else,it is recommended that you now start indexing structures by using Smart Discovery Indexing.
			For structures indexed using Smart Discovery Indexing, supports: • Advanced find and	For structures indexed using Smart Discovery Indexing, does not support: • Massive Model Visualization	Index structures by using Smart Discovery Indexing.

Search type	Supported on	Description	Supports	Does not support	Required configuration
			find by saved queries • Visualization of filtered structures • Volume and proximity filters • Advanced configuration of a structure, such as configure by selection and configure by proximity. Additionally, the filtered structure can be saved, replayed, interoperated, or shared with other users.	• Classification search	

Configure spatial searches for structures

The following installation and configuration steps must be completed before you can run spatial searches:

The following installation and configuration steps must be completed before you can run spatial searches:

- Install the **Spatial Search** feature when you run Teamcenter Environment Manager (TEM).
- If you have not already done so, install Dispatcher translation services with Teamcenter Environment Manager (TEM), including the **Spatial Search Indexer** and **Spatial Search Translator** features. If you are creating JT files on the same system as Teamcenter is installed on, ensure you install these services before creating JT files to search.
- Install a license key that includes an RDV (Context Management) license.
- To enable the cacheless search mechanism, set the **QS_QSEARCH_ENABLED** site preference to **True**. If you use appearance searches, set this preference to **False**.
- To enable spatial (box or proximity) searches, set the **QS_SPATIAL_ENABLED** site preference to **True**.
- If you search against TruShape files that Teamcenter creates from CAD data, you must also set the **QS_TRUSHAPE_GENERATION_ENABLED** site preference to **True**. This enables the generation of the TruShape data required for spatial searches; the generation process may take an appreciable time when first enabled. Set this preference to **False** if you use the Teamcenter Integration for NX, as TruShape files are automatically saved in the NX datasets. (NX must also be configured to save TruShape data.)
- If you search against existing JT files, you must also set the **QS_BBOX_GENERATION_ENABLED** site preference to **True**. This enables the generation of the bounding box data required for spatial searches; the generation process may take an appreciable time when first enabled. Set this preference to **False** if you use the Teamcenter Integration for NX, as bounding boxes are automatically generated. However, set this preference to **True** if you use non-NX CAD integrations or several CAD tools.
- If you are upgrading from a previous release of Teamcenter, use the **create_or_update_bbox_and_tso** utility to convert your existing data for spatial searching. This utility creates bounding box and TruShape data from CAD data files or JT files.
- To specify the unit of measure for a proximity search in a Smart Discovery-indexed structure, set the **RDV_user_defined_units_of_measure** site preference to **METERS**, **MILLIMETERS**, **INCHES**, or **UNKNOWN**. If you specify **UNKNOWN**, the search is performed in the units of the **assy_units** property set on the top level BVR. If this property is not set, the search is performed in the units specified in the **PS_assume_legacy_transform_units** site preference.
- Set the **StructureManagerIncludeSubComponentsForSpatialSearch** site preference to determine if the components of subassemblies are considered as targets. If it is set to **true**, all components of the selected subassembly are included as targets of a proximity search, in addition to the subassembly itself. If it is set to **false**, only the subassembly is the target of a proximity search, not any of its components. Typically set to **false** if you want to search for parts in the vicinity of geometry at the assembly level (for example, in the case of wire harnesses).

Note:

If you change the settings of any of these site preferences, you must restart the server to clear cached data and restart the services.

13. Configure sessions

About configuring sessions

Users can save the filtered and configured structures in *sessions*, which help them return to the product definitions that they are currently working with. You can configure a session so that:

- It **loads** either with static data or with the latest data.
- Users can **share** their sessions with other users.
- One user cannot **overwrite** the changes made by the other user when two users work with the same session.

Set how data must be loaded in a session

Users can save the filtered and configured structures in a session. You can choose whether to load a session with the static data stored in the session or with the latest data. To do this, you edit the **DesignContextLoadRDVContextObjectMode** preference. Set its value to:

- **0** if you want to load the session with the data already stored in it.
- **1** (default) if you want to load the session with the latest data. In this case, the stored data is re-evaluated against the current data, and the latest data is loaded in the session.

Change the default sharing behavior of a session

Users can share a session with other users for collaboration. The default sharing behavior of a session is set in the **Has Class(Fnd0AppSession)** Access Manager rule. This rule is available under **Has Class(POM Object)** > **Has Class(POM_application_object)**.

The **Fnd0AppSession** rule has three values:

- Private (**PrivateAppSessionACL**) — only the owning user can view or modify sessions. Other users cannot view the sessions.
- ReadOnly (**ReadOnlyAppSessionACL**) — only the owning user can modify sessions. Other users can view the sessions.
- ReadWrite (**ReadWriteAppSessionACL**) — all users can view or modify sessions.

To change the default sharing behavior of a session, you must modify these values.

Restrict users from overwriting sessions

A user can overwrite a session that is opened by another user for making updates. As an administrator, you can restrict users from overwriting sessions. To do so, you set the **AWBShowOverwriteForConcurrentSave** preference to **false**.

14. Configure worksets to use arrangements

You can allow users to configure a structure in a workset using arrangements by setting the **AW_Allow_Arrangement_On_Worksetchildren** preference to **true**. If you set this preference to **true**, the users can use the **Arrangement** list in a workset and then configure the structure.

By default, this preference is set to **false**.

15. Specify structures as precise or imprecise

In Active Workspace, you can specify if a structure is precise or imprecise by editing the top line of the structure from the **Properties** panel and selecting the **Precise** property. The default value is **Imprecise**.

If you do not see the **Precise** property for an element such as a logical block, you must edit the XRT of that element and add the following property under the **tc_xrt_Properties** section:

```
<property name="awb0IsPrecise">
```

> Note:
>
> To set the default value for newly created product structures as precise, change the value of the **TC_BOM_Precision_Preference** preference to **Precise**. (The default value is **Imprecise**.)

16. Administer working contexts for Active Workspace

Enable the sharing of a saved working context

Users can save a working context to share with other users, stakeholders, and collaborators. To enable the sharing of a user's saved working context, the system administrator must add the **Has Class(Awb0SavedBookmark)** rule to the Access Manager (AM) rule tree.

You can add the rule using the rich client. Rules are evaluated based on their placement in the tree structure. Therefore, the location of the rules in the rule tree is important.

1. Add the **Has Class(Awb0SavedBookmark)** rule and set the **Condition**, **Value**, and **ACL Name** as follows:

```
Has Class( Awb0SavedBookmark )
Has Attribute( Awb0SavedBookmark:awb0ShareLevel=Private )-
>PrivateSBMACL
Has Attribute( Awb0SavedBookmark:awb0ShareLevel=Read Only )-
>ReadOnlySBMACL
Where:
PrivateSBMACL:    Owning User: Grant Read, Write, Export
                  World: Deny Read, Write, Export
ReadOnlySBMACL:   Owning User: Grant Write
                  World: Deny Write
```

2. Click **Add** and then click **Save**.

Clean up background working contexts

Teamcenter deletes background working contexts to free up disk space, according to their age and the maximum number of allowed contexts per user. To control this clean up, set the following preferences:

- **AWBBackgroundContextCleanupDays**

  Controls the maximum time working contexts are kept. For example, if this preference is set to 30 days, when a user opens a structure, all working contexts older than 30 days are deleted.

- **AWBBackgroundContextMaxCountPerUser**

  Controls the maximum number of working contexts kept for each user. For example, if this preference is set to 50, when a user opens a structure, the oldest working contexts from the set of that user's contexts beyond the limit of 50 are deleted.

The minimum valid value for the **AWBBackgroundContextMaxCountPerUser** preference is **1**. The users must set this preference to **1** or more.

These settings applies to all users and it is not possible to set a different cleanup period for individual users.

17. Administer revision rules

About revision rules

A *revision rule* sets the criteria for selecting the appropriate revision of elements in a product structure. For example, users can choose whether to load working revisions or release revisions. As a privileged user (or an administrator), you set up revision rules by defining *revision rule entries*. Each rule entry defines a parameter that is used for evaluating which structure configuration must be loaded. The revision rule is evaluated based on the order of precedence of the rule entries. As soon as a revision is successfully configured, the rest of the rule entries are not evaluated. Therefore, you must set the order based on your site requirement.

The different rule entries are:

Revision rule entry	Description	Example

Working	It is used for loading the working item revision that do not have a release status. By default, the latest working revision is selected according to the date it was created.
	It can be made more specific by specifying an owing user and owning group.
		To load elements that John or Project X group are currently working on, you set up the revision rule as:
		`Working( Owning User = John )`
		`Working( Owning Group = Project X)`
		To load elements that belongs to the user currently logged on:
		`Working( Owning User = Current )`
		To load elements that belong to the group to which the user is currently logged on to:
		`Working( Owning Group = Current )`

Status	It is used for loading item revisions that are released with a specific status. It can be made more specific by specifying a specific status, release date, effective date, and effective unit number.	To load the latest revision of elements as per their effective production date: `Has Status (Production, Configured by Effective Date )` To load the most recently released revision of elements that are either in the **Production** or **Pre-Production** state:

Revision rule entry	Description	Example
		`Has Status( Production, Configured by Date Released ) Has Status( Pre-Production, Configured by Date Released)`

Latest	It is used for loading item revisions regardless of whether they are released or not. It can be made more specific by including the creation date, alphanumeric revision ID, numeric revision ID, and a combination of alpha and number revision IDs.	To load the latest revisions of elements based on their creation date:
		`Latest( Creation Date )`
		To load the latest revisions of elements based on numeric revision ID:
		`Latest( Numeric Rev ID)`
		Here, elements with non-numeric revision IDs are not loaded.

Date	It is used for loading item revision with a specific date. You can use this entry only with other entries.	To load the latest revision of elements as on today:
		`Latest, today`
		To load the latest revision of elements that are in **Production** as of **1-Jan-2007**:
		`Has Status( Production, Release Date = 1-Jan-2007)`

Unit Number	It is used in combination with the status rule entry with unit number effectivity. It is used for loading item revisions with a unit number as specified by the user. Typically, a unit number is a property of the end product or a major module of a product. As Teamcenter may manage many units, you typically qualify a unit number entry with an end item entry.	To load the element revisions that are in **Production** based on a specific unit number: `Has Status( Production, Configured by Unit Number )`
End item	It is a product, system, or module with respect to which you configure a structure by effectivity. For example, you can configure the structure of unit number **110** in product **X400**, where **X400** is the end item.	

Revision rule entry	Description	Example
Precise	It is used for loading item revisions in a precise product structure.	
Override	It is used for overriding all item revisions available in an **override folder**. The revision rule will not be evaluated for these item revisions.	

You can modify the order of the rule entries to change the precedence used when evaluating the revision rule. Certain rule entries can also be **grouped** so they are evaluated with equal precedence. In the case of status entries, you can group two or more different statuses with equal precedence. In the case of working entries, you can group different owners with equal precedence.

> Example:
>
> To load recently released revisions of structure elements irrespective of whether they are in the **Production** or **Pre-Productions** state, you group the status rule entry as follows:
>
> ```
> [ Has Status( Production ), Configured by Date Released )
> Has Status( Pre-Production, Configured by Date Released )]
> ```

You can also group entries according to **item type** and **occurrence type**. Certain entries cannot be grouped together. The following are not valid combinations for grouping:

- ```
  Latest
  Has Status, Configured By Release Date
  ```

- ```
  Has Status( TCM Released ), Configured by EffectiveDate
  Has Status( Pending ), Configured by Release Date
  ```

Revision rule entries: Scenarios

Working entry: Scenario

A *working entry* is used for selecting working item revisions that do no have a release status. By default, the latest working revision is selected according to the date it was created. It can be made more specific by specifying an owing user and owning group.

> Tip:
>
> There may be more than one working entry within a revision rule. For example, a rule may configure the current user's working revision and, if none is found, configure the current group's

working revision instead. If a user changes group, it is necessary to reapply the revision rule to configure the appropriate revisions for the new group.

However, in many circumstances, it is good practice to limit the number of working revisions, typically to a single revision. Your Teamcenter administrator can do this in Business Modeler IDE.

Example of a Working entry

```
Working( Owning User = John )
Working( Owning Group = Project X )
```

Assume you configure the following two items with this rule:

```
Part1/A : Status = Production
Part1/B : Working, Owning User = John, Owning Group = Project Y
Part1/C : Working, Owning User = Jane, Owning Group = Project X
Part2/A : Status = Production
Part2/B : Working, Owning User = Jane, Owning Group = Project X
```

Teamcenter configures Revision B of Part 1, because it is owned by John. The owning groups are not relevant.

Teamcenter configures Revision B of Part 2, because it is owned by Project X. There is no revision owned by John.

Note:

If John or Project X owned more than one revision, Teamcenter would configure the latest created revision of the matching revisions.

Example of a Working entry with current user/group

```
Working( Owning User = Current )
```

Assume you configure the following item with this rule:

```
Part1/A : Status = Production
Part1/B : Working, Owning User = John, Owning Group = Project Y
Part1/C : Working, Owning User = Jane, Owning Group = Project X
Part1/D : Working, Owning User = Jane, Owning group = Project Y
```

This rule configures a revision that depends on the identity of the *user* logged into Teamcenter.

- If Jane is logged in, Teamcenter configures Revision D.
- If John is logged in, Teamcenter configures Revision B.

Assume you use the following rule to configure the same item:

```
Working( Owning Group = Current )
```

The revision configured by this rule is dependent on which group the user is logged into Teamcenter.

This rule configures a revision that depends on the *group* in which the user logged into Teamcenter.

- If the user is logged into Group Project X, Teamcenter configures Revision C.
- If the user is logged into Group Project Y, Teamcenter configures Revision D.

Status entry: Scenario

A status entry is used for selecting item revisions that are released with a particular status. The following settings are available for status entries:

Any release status	Teamcenter configures the latest item revision with a released status, regardless of the actual status.
Selected status	Teamcenter configures the latest item revision with a selected status type. This setting allows you to configure a structure that contains only item revisions with a specified status.
Released date	Teamcenter selects the latest item revision according to the date the revision was released (that is, the date the particular status was added).
Effective date	Teamcenter selects the latest item revision according to effectivity dates defined on the release status.
Effective unit number	Teamcenter selects the latest item revision according to unit numbers defined on the release status.

Example of status hierarchy in a revision rule

The following example shows how to use status hierarchy in a revision rule:

```
Has Status( Production, Configured by Date Released )
Has Status ( Pre-Production, Configured by Date Released )
```

Assume you configure the following item with this rule (dates are release dates):

```
Part1/A : Status = Pre-Production [1-Apr-2007]
Part1/B : Status = Pre-Production [1-Jun-2007]
Part1/C : Status = Production [1-Aug-2007]
Part1/D : Working
Part2/A : Status = Pre-Production [1-May-2007]
```

```
Part2/B : Status = Production [1-Jul-2007]
Part2/C : Status = Pre-Production [1-Sep-2007]
Part3/A : Status = Pre-Production [1-Aug-2007]
Part3/B : Working
```

Teamcenter configures Revision C of Part 1 because it is the most recently released revision with **Production** status.

It configures Revision B of Part 2 because it is also the most recently released revision with **Production** status. The later preproduction revision is not configured.

It configures Revision A of Part 3. There is no revision with **Production** status, so it configures the latest **Pre-Production revision**.

> Note:
>
> The rule in this example creates a status hierarchy. If possible, the rules configures a **Production** release, but if one is not available, it configures a **Pre-Production** release.

Example of effective date configuration in a revision rule

The following example shows effective date configuration in a revision rule:

```
Has Status( Production, Configured by Effective Date )
```

Assume you configure the following item with this rule (dates are effective date ranges):

```
Part1/A : Status = Production [1-Apr-2007 to ... ]
Part1/B : Status = Production [1-Aug-2007 to ... ]
Part1/C : Status = Production [1-Nov-2008 to ... ]
```

The release status has both start date and end date attributes, but typically the end date attribute is not populated. The system selects the latest effective revision of the appropriate status type with a start date before the date specified in the date revision rule entry.

If no date is set in the rule, it defaults to today, and the revision with the latest start date is selected.

In the previous example, if today's date is sometime in 2007, Teamcenter configures revision B, because it has **Production** status and also has the later effective start date of the two revisions effective in 2007.

If no start date or end date are defined on the release status of an item revision, it is assumed to be not effective; that is,effectivity criteria must be defined for an item revision to become effective,

Example of effective unit number configuration in a revision rule

The following example shows effective unit number configuration in a revision rule:

```
Has Status( Production, Configured by Unit Number )
```

Assume you configure the following item with this rule:

```
Part1/A : Status = Production [3 to 10]
Part1/B : Status = Production [11 to 15]
Part1/C : Status = Production [16 to UP]
```

- If the unit number is set to 3, Teamcenter configures revision A. Revision A falls within the range 3 to 10, but neither of the other ranges.
- If the unit number is set to 11, Teamcenter configures revision B. Revision B falls within the range of 11 to 15 upward, but neither of the other ranges.
- If the unit number is set to 16, Teamcenter configures revision C. Revision C falls within the range of 16 upward, but neither of the other ranges.
- If the unit number is set to 2, no revision of Part1 is configured. Unit number 2 is outside the effectivity ranges of all three parts.

The release status has both start unit and end unit attributes, and typically both attributes are populated. The system selects the revision of the appropriate status type with the start unit closest to, and before, the unit number specified in the unit number revision rule entry. If no unit number is set in the rule, there is no practical default and no revisions are selected

If no start unit or end unit are defined on the release status of an item revision, it is assumed to be not effective. That is, the effectivity criteria must be defined on an item revision to make it effective.

Split unit number ranges can be defined by attaching multiple release statuses to the status type.

If multiple revisions are eligible for configuration, the one with the highest start unit number is selected.

Latest entry: Scenario

A *latest* entry is used for selecting the latest item revisions regardless of whether they are released or not. For this entry, Teamcenter does not differentiate between working revisions and revisions with status. The following settings are available for **Latest** entries:

Creation Date	Selects the latest item revision by the date the revision was created.
Alphanumeric Rev ID	Selects the latest item revision by revision ID in alphanumeric (individual characters) order.

Numeric Rev ID	Selects the latest item revision by revision ID in numeric order. Revisions with non-numeric IDs are not configured.
Alpha+Number Rev ID	Selects the latest item revision by revision ID, sorting letter portions as an alphabetic group, and sorting digit portions as a number.

For example:

Setting	Rev ID 1	Rev ID 2	Rev ID 3	Latest	Notes
Alphanumeric Rev ID	aa	b		b	Letters sorted alphabetically (individual letters). ANSI value of b is > a.
Alpha + Number Rev ID	aa	b		aa	Letters grouped and sorted numerically. ANSI value of aa is > b.
Alphanumeric Rev ID	21	3		3	Digits sorted as alphabetic (individual digits). ANSI value of 3 is > 2.
Alpha + Number Rev ID	21	3		21	Digits grouped and sorted as number. 21 is > 3.
Alphanumeric Rev ID	a5	b3	b23	b3	ANSI value of b > a, so on first character comparison, b3 and b23 are selected. Continuing to the second character, ANSI value of 3 > 2.
Alpha + Number Rev ID	a5	b3	b23	b23	ANSI value of b > a, so on first character comparison b3 and b23 are selected. Continuing, numerical digits are grouped and compared by number value.

Setting	Rev ID 1	Rev ID 2	Rev ID 3	Latest	Notes
					23 > 3.
Alphanumeric Rev ID	aa4	bc123	bc22	bc22	ANSI value of b > a, so on first character comparison bc123 and bc22 are selected. Continuing, as the first two characters are the same it goes for third character where ANSI value of 2 > 1.
Alpha + Number Rev ID	aa4	bc123	bc22	bc123	ANSI value of bc > aa, so on first comparison bc123 and bc22 are selected. Continuing, numerical digits are grouped and compared by number value. 123 > 22.

Example of a revision rule with latest by creation date entry

The following example shows a revision rule that configures with the latest by creation date:

```
Latest( Creation Date )
```

If you configure the following item with this rule (dates are creation dates):

```
Part1/A : Status = Production [1-Apr-2006]
Part1/B : Working [1-Jun-2006]
```

Teamcenter configures Revision B because it was created later than revision A. It is not relevant if the revision is working or has status.

Date entry: Scenario

A date entry is used for specifying a date to configure the structure against. You can only use this entry type with other entries. These types of entry find the latest entry before the specified date:

- Status entry – released date or effective date
- Latest entry – creation date

You can set the date in a date entry to **Today**, and Teamcenter evaluates the configuration criteria against the current date and time.

You cannot configure working revisions against a date in the past. Teamcenter does not maintain information about the revisions that were in a working state at a particular time in the past.

> Note:
>
> If no date entry is present in a revision rule, Teamcenter evaluates the date by default to today's date.

Grouped entries: Scenario

You can group status entries and working entries for equal precedence. In the case of status entries, you can group two or more different statuses with equal precedence.

Example of revision rule with grouped entries

```
[ Has Status( Production, Configured by Date Released )
Has Status( Pre-Production, Configured by Date Released ) ]
```

Assume you configure the following three items with this rule (dates are release dates)

```
Part1/A : Status = Pre-Production [1-Apr-2007]
Part1/B : Status = Production [1-Jun-2007]
Part1/C : Status = Production [1-Aug-2007]
Part1/D : Working
Part2/A : Status = Pre-Production [1-May-2007]
Part2/B : Status = Production [1-Jul-2007]
Part2/C : Status = Pre-Production [1-Sep-2007}
Part3/A : Status = Pre-Production [1-Aug-2007]
Part3/B : Working
```

Teamcenter configures Revision C of Part 1, because it is the latest released revision with either **Production** or **Pre-Production** status.

Teamcenter configures Revision C of Part 2 for the same reason.

Teamcenter configures Revision A of Part 3, again for the same reason.

Precise entry: Scenario

If an assembly is precise and the revision rule contains a **precise** entry, the assembly is configured according to that entry. However, if the assembly is precise and the revision rule does *not* contain a **precise** entry, the assembly is treated as imprecise.

For example:

- You have an item called Part1 with two revisions, Part1/A is **Working** and Part1/B is also **Working**.

  A precise assembly **Asm** is created containing Part1/B. That is:

  ```
  Asm
   |--Part1/B
  ```

  If User2 is denied read access to Part1/B and opens **Asm** in Structure Manager with an **AnyStatus;Working** revision rule, the system performs the following evaluation:

  1. The **AnyStatus** entry does not configure any revisions.
  2. The **Working** entry configures in Part1/B but access rules make this revision **<UNREADABLE>**.
  3. Regardless of the setting of the **PSE_check_read_access_for_Rev_Rule_entry** preference, evaluation continues. The next revision is Part1/A, and access rules do not deny access to it, so the final result is Part1/A.

  If User2 then changes the revision rule to **Latest by Alpha Revision Order**, the system performs the following evaluation:

  1. The **Latest by Alpha Revision** entry configures in Part1/B but access rules make this revision **<UNREADABLE>**.
  2. The system continues evaluation only if the **PSE_check_read_access_for_Rev_Rule_entry** preference is **TRUE**. The next revision is Part1/A, and access rules do not deny access to it, so the final result is Part1/A.

- You have an item called Part1 with three revisions, Part1/A is released with a **TcReleased** status, Part1/B is released with a **TcBaseline** status, and Part1/C is released with a **TcReleased** status.

  A precise assembly **Asm** is created with Part1/C. That is:

  ```
  Asm
   |--Part1/C
  ```

  If User2 is denied read access for the **TcReleased** status and opens **Asm** in Structure Manager with a **AnyStatus;Working** revision rule, the system performs the following evaluation:

1. The **AnyStatus** entry configures in Part1/C but access rules make this revision **<UNREADABLE>**.

2. The system continues evaluation only if the **PSE_check_read_access_for_Rev_Rule_entry** preference is **TRUE**. The next revision with status is Part1/B, and access rules do not deny access, so the final result is Part1/B.

- You have an item called Part1 with two revisions, Part1/A is released with a **TcReleased** status and Part1/B is working.

  A precise assembly **Asm** is created with Part1/A. That is:

  ```
  Asm
   |--Part1/A
  ```

  If User2 is denied read access for status **TcReleased** and opens **Asm** in Structure Manager with a **AnyStatus;Working** revision rule, the system performs the following evaluation:

  1. The **AnyStatus** entry configures in Part1/A but access rules make this revision **<UNREADABLE>**.

  2. The system continues evaluation only if the **PSE_check_read_access_for_Rev_Rule_entry** preference is **TRUE**. Because no other revisions have status, the next rule entry of **Working** configures in Part1/B. Access rules do not deny access, so the final result is Part1/B.

  If User2 changes the revision rule to **Latest Working**, the system performs the following evaluation:

  1. The **LatestWorking** rule contains a **Precise** entry, that configures the revision saved with the assembly, namely, Part1/A.

  2. Because access rules make Part1/A unreadable for User2, the final result is **<UNREADABLE>**.

     The **PSE_check_read_access_for_Rev_Rule_entry** preference does not affect the result because the structure and the revision rule are precise.

Revision rules for incremental changes: Scenario

Review the example of how the **Pre-Released** status can be applied to the item revisions and the **IC in Process** status to the incremental change revisions.

The revision rule needed for that example is as follows:

Has Status = IC in Process, Configured by Unit:
Has Status = Pre-Released, Configured by Unit:
Has Status=Released, Configured by Unit
Has Status = Released, Configured by Release Date
Has Status = Pre-Released, Configured by Release Date
Working

The purpose of each entry in this revision rule is as follows:

Has Status = IC in Process, Configured by Unit:
Has Status = Pre-Released, Configured by Unit:

Allows Teamcenter to configure both the incremental change and the structure revisions by unit number. Having separate statuses allows the application of different access controls to the item revision and BOM view revision and to the incremental change.

Has Status=Released, Configured by Unit

Ensures the upper levels of the structure are configured correctly by standard revision configuration, as well as any finally released revisions at lower levels.

Has Status = Released, Configured by Release Date
Has Status = Pre-Released, Configured by Release Date
Working

Configure item revisions that do not have unit effectivity applied or are unreleased, that is, **working**.

Group revision rule entries in the rich client

Group revision rule entries by item type

You can modify the order of precedence of revision rule entries by grouping revision rules by item type. Teamcenter evaluates any revision rule under the item type group for configuration of item revision only if the item type of the revision rule group and underlying item revision matches. By using this grouping mechanism in a product structure that has item revisions of different item types, you can create revision rules that selectively apply the revision rule entries according to the given item type.

For example, you may use an item to manage parts and equipment, but need to distinguish between a production part and the equipment used to manufacture the part. Each of these item types may have the same status but can be configured in a different way, for example, by unit or date released. To achieve this, you can define a revision rule with a clause that restricts some of the entries to certain item types. To do so, create a **Has Item Type** entry in a revision rule in the following format:

```
Has Item Type (<item type 1, item type 2, etc) {

<Entry 1>

<Entry 2>
etc.}
```

Example:

```
Has Item Type (Part) }
 Has status (Approved, Configured with Released Date) }
 Has status (Production, Configured with Unit Number)
```

Here the **Has status (Approved, Configured with Released Date)** revision rule entry configures the product structure only if the item type is **Part**. For all other item types in the product structure, Teamcenter applies the **Has status (Production, Configured with Unit Number)** revision rule entry. If neither of these entries applies, the part is not configured by this revision rule.

Grouping entries for multiple item types

Each **Has Item Type** entry may have one or more item type arguments, but cannot be left blank, that is, you cannot create a revision rule entry for **Has Item Type (Any)**. To include more than one item type argument, use the syntax in the following example:

```
Has Item Type ( Prototype, Virtual Build ) {
...}
```

Grouping combinations of item type entries

You can only use one level of grouping and may not create nesting inside an existing group entry, as shown in the following examples:

- Single revision rule entry grouped for item type:

```
Has Item Type ( Prototype ) {
  entry 1 }
entry2
```

- Multiple revision rule entries grouped for item type without equal precedence:

```
Has Item Type ( Prototype ) {
  entry 1
  entry 2 }
entry3
```

- Multiple revision rule entries grouped for item type with and without equal precedence:

```
Has Item Type ( Part ) {
  [Status (Released, Configured Using Release Date) }
Has Item Type ( Module, Incremental Change ) {
  [Status (Released, Configured Using Unit No) }
Working
```

This example shows how you can use the same status type in both situations, but configured differently—in this case, by date for parts, but by unit number and incremental change for modules.

You can only group **Working** and **Status** entries with equal precedence.

Group revision rule entries by occurrence type

You, as a configuration management analyst, can define revision rules to treat different occurrence types with variations of the configuration logic. In some cases, the existing grouping mechanisms based on the item type are not sufficient because a part appears in some assemblies only for reference purposes. In such cases, you can use a **Working revision** rule to author the BOM significant parts in an assembly. However, the system may reference the content, using a **Released revision** rule. For example, you create a **Has Occurrence** for the **MEAssign** occurrence type. Then you create a **Working()** revision rule to configure the product structure only if the occurrence type is **MEAssign**. For all other occurrence types in the product structure, Teamcenter applies the **Has Status(Any Released Status, Configured Using Released Date)** rule.

You can modify the order of precedence of revision rule entries by grouping revision rules by occurrence type. For the configuration of an item revision, Teamcenter evaluates a revision rule within the occurrence type group only if there is a match between the occurrence type of the revision rule group and the underlying item revision on the line. By using this grouping mechanism with item revisions of different occurrence types, you can create revision rules to selectively apply the revision rule entries according to a given occurrence type.

You cannot create item type groups within occurrence type groups.

You create the **Has Occurrence** type in the following format:

```
Has Occurrence Type(Occurrence type1, Occurrence type 2, etc) {
<Revision Rule Entry 1>
<Revision Rule Entry 2>
……
etc. }
```

Example:

```
Has Occurrence Type(MEAssign) {
Working()  }
Has Status(Any Release Status, Configured Using Released Date)
```

In this example, the **Working()** revision rule entry configures the product structure only if the occurrence type is **MEAssign**. For all other occurrence types in the product structure, Teamcenter applies the **Has Status(Any Released Status, Configured Using Released Date)** rule.

Group combinations of occurrence type entries

You can only use one level of grouping as shown in the following examples.

Do not create nested entries within an existing group entry.

- Single revision rule entry grouped for occurrence type:

```
Has Occurrence Type ( OccurrenceType1 ) {
  entry 1 }
entry2
```

- Multiple revision rule entries grouped for occurrence type without equal precedence:

```
Has Occurrence Type ( OccurrenceType2 ) {
  entry 1
  entry 2 }
entry3
```

- Multiple revision rule entries grouped for occurrence type with and without equal precedence:

```
Has Occurrence Type ( MEAssign ) {
  [Status (Released, Configured Using Release Date) }
Has Occurrence Type ( MeAssemble ) {
  [Status (Released, Configured Using Unit No) }
Working
```

This example shows how you can use the same status type for both situations by configuring it differently. In this case, you use the by date for **MEAssign** and the by unit number for **MeAssemble**.

Group existing revision rule entries by item type

1. Choose **Tools→Revision Rule→Create/Edit**.

   Teamcenter displays the **Revision Rule** dialog box.

2. In the dialog box, select one or more revision rule entries and click the **Group** button.

   Teamcenter displays the **Group Entries** dialog box.

3. In the dialog box, click the **Group Entries by Item Types** button or the **Group Entries by Equal Precedence** button, then select the item types you want to group in the **Available Item Types** list. You can transfer the selected item types to the **Configure By→Has Item Type** box in the dialog box or remove them by clicking the **+** or **–** buttons respectively. Click **OK** when you have grouped the entries.

   Teamcenter updates the selected item types in the **Configure By→Has Item Type** box of the dialog box, with a partially created **Has Item Type** revision rule entry.

> Note:
>
> You can only group **Working** and **Status** entries with equal precedence.

Ungroup combinations of item type entries

1. Choose **Tools→Revision Rule→Create/Edit**.

   Teamcenter displays the **Revision Rule** dialog box.

2. In the dialog box, select one or more **Has Item Type** revision rule entries that are grouped by item type, and click the **Ungroup** button.

   Teamcenter removes the selected entries and updates the dialog box.

Create a revision rule

How to create revision rules

If you are a privileged user, you can use two dialog boxes to create or edit a revision rule:

- The **Modify Current Rule** dialog box to create temporary revision rules.
- The **Create/Edit Rules** dialog box to create persistent revision rules.

Both dialog boxes contain the Revision Rule Editor, which comprises two main panes.

- The upper pane lists all the entries in the rule, with buttons you can use to manipulate the entries.
- The lower pane allows you to create or edit an entry, and add it into the rule.

Set an override folder

To add an override folder into the currently active revision rule, choose **Tools→Revision Rule→Set Override Folder**. This command is available to unprivileged users, but is only active for revision rules that contain a special empty override entry (no folder specified). Thus privileged users who define the revision rules should place an empty override entry into a revision rule if they want to allow unprivileged users to use their own override folders with the rule.

The **Set Override Folder** dialog box allows you to set an override folder and also to see the folder that is currently set. The dialog box has the following boxes:

- A **Name** box into which you can enter a case sensitive search string to find the required override folder. Click the **Find**  button next to this box to perform the search.

- The search results box. Double-click the folder you want to set.
- A **Folder** box that displays the folder currently used as the override folder in the revision rule.

To use the existing override folder, simply click **OK** or **Cancel**.

Note:

- The first time you make a change (such as setting an override) to a saved revision rule, Teamcenter creates a modified copy of the rule and applies it to the window. This is the rule to which you then make the override change.
- When you make a change to a folder (for example, adding an item revision to it when Teamcenter Integration for NX is active) and open an assembly configured by a rule that uses the override folder, Teamcenter does not automatically reevaluate the revision rule. Consequently, it does not take into account the change to the override folder. This rule is reevaluated only if you choose **File→Options→Load Options** in Teamcenter Integration for NX. Click **OK** before opening the part file.

Create a rule entry

1. In the Revision Rule Editor, select the type of entry you want to create. By default, the combination box shows **Working**. Teamcenter updates the entry boxes appropriately.
2. Enter values into the boxes and click one of the following buttons:
   - Click **Append** to add your entry at the end of the list of rule entries.
   - Click **Insert** to add your entry above the selected entry.
   - Click **Replace** to replace the selected entry with your entry.

Create a revision rule for nested effectivity

When you create a revision rule for a nested effectivity scheme, ensure you select the **Nested Effectivity** check box. This action adds a **Nested Effectivity** entry to the revision rule and ensures that Teamcenter modifies the revision rule according to the mapping on any configuration items it encounters. If you do not check this check box, Teamcenter ignores any configuration items. The position of the **Nested Effectivity** entry in the rule is not significant and never changes.

The following example structure has date effectivity and an end item defined on the item revision of each part:

Item	Status, effectivity	Note
NE-A1000/A (CI) – Product X		
NE-A2000/A (CI)	Released, EI: NE-A1000, 15 July-UP	Rev Effect wrt NE-A1000 – Product X
NE-A100/A	Released, EI: NE-A2000, 20 July-UP	
NE-A200/A	Released, EI: NE-A2000, 20 July-UP	
NE-PP30/A	Released, EI: NE-A2000, 20 July-UP	
NE-PP10/A	Released, EI: NE-A2000, 20 July-UP	
NE-PP20/A	Released, EI: NE-A2000, 20 July-UP	
NE-A3100/A (CI)	Released, EI: NE-A1000, 16 July-UP	Rev Effect wrt NE-A1000 – Product X
NE-A400/A	Released, EI: NE-A3100, 23 July-UP	
NE-PP70/A	Released, EI: NE-A3100, 25 July-UP	
NE-PP60/A	Released, EI: NE-A3100, 25 July-UP	
NE-A5000/A (CI) – Product Y		
NE-A2000/A	Released, EI: NE-A5000, 15 July-UP	Rev Effect wrt NE-A5000 – Product Y

The resulting configuration item effectivity mappings are:

- Mapping on NE-A2000 (CI):

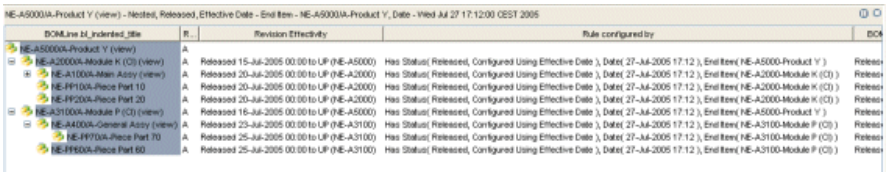
  NE-A1000 (CI) – Product X : NE-A2000 (CI)
  NE-A5000 (CI) – Product Y : NE-A2000 (CI)

- Mapping on NE-A3100 (CI):

  NE-A2000 (CI) – Product Y : NE-A3100 (CI)

If you apply the following revision rule, the structure is configured as shown in the following figure:

Status = Released, Configured by Effective Date
Working
Nested Effectivity
EI = NE-A5000
Date = 27 July 2006



Nested effectivity 1

Conversely, if you apply the following revision rule with no nested effectivity entry, the structure is configured as shown in the following figure:

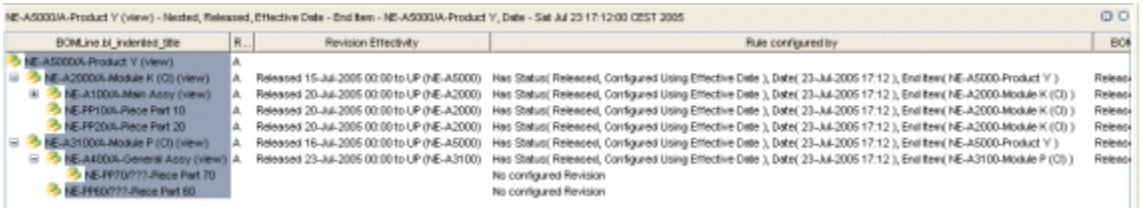
Status = Released, Configured by Effective Date
Working
EI = NE-A5000
Date = 16 July 2005

Nested effectivity 2

Teamcenter ignores the mappings, so the end item does not switch from A5000 to NE-A2000/NE-A3100 at the configuration items. Consequently, none of the components below the configuration items are configured, as the end item on the effectivity is not NE-A5000.

If you apply the following revision rule, Teamcenter applies effectivity mapping due to the **Nested Effectivity** entry and the structure is configured as shown in the following figure. However, some of the components are not effective on 23 July and have no configured revision; Teamcenter marks these components as **???**.

Status = Released, Configured by Effective Date
Working
Nested Effectivity
EI = NE-A5000
Date = 23 July 2005



Nested effectivity 3

Create and group revision rule entries by equal precedence

1. Choose **Tools→Revision Rules→Create/Edit**.

   Teamcenter displays the **Revision Rules** dialog box.

2. In the dialog box, click the **Create/Edit** button.

   Teamcenter displays the **Revision Rule** dialog box with any existing revision rule entries.

3. In the dialog box, define and select the entries of the same type you want to group and click the **Group** button.

   Teamcenter displays the **Group Entries** dialog box.

4. In the dialog box, click the **Group Entries by Equal Precedence** button.

   Click **OK** and Teamcenter updates the **Revision Rule** dialog box with the newly grouped revision rule entries.

   > Note:
   >
   > When grouping by equal precedence, you should only group **Working** entries with other **Working** entries and **Status** entries with other **Status** entries. Do not group other combinations of entries.

5. In the **Revision Rule** dialog box, complete the definition of the revision rule and click **OK**.

> Note:
>
> To remove an equal precedence entry from a revision rule, you must highlight *all* the lines of the entry in the dialog box.
>
> You can also group revision rule entries using engineering change types.

Create and group Has Item Type revision rule entries

1. Choose **Tools→Revision Rules→Create/Edit**.

   Teamcenter displays the **Revision Rules** dialog box.

2. In the dialog box, click the **Create/Edit** button.

   Teamcenter displays the **Create New Revision Rule** dialog box with any existing revision rule entries.

3. In the dialog box, define and select the entries you want to group and click the **Group** button.

   Teamcenter displays the **Group Entries** dialog box.

4. In the dialog box, click the **Group Entries by Item Types** button, and then select the item types you want to group in the **Available Item Types** list. You can transfer the selected item types to the **Configure By→Has Item Type** box in the dialog box or remove them by clicking the **+** or **–** buttons, respectively.

   Click **OK** to update the selected item types in the **Configure By→Has Item Type** box of the dialog box, with a partially created **Has Item Type** revision rule entry. It also updates the **Revision Rule** dialog box with the newly grouped revision rule entries.

5. In the **New Revision Rule** dialog box, complete the definition of the revision rule and click **OK**.

Modify a rule entry

1. Select an entry in the list of entries.

2. Click **Copy**. Teamcenter copies the entry into the editing area and displays the appropriate entry type and boxes.

3. Edit the values of the entry and click **Replace** with the original entry still selected. Teamcenter replaces the old entry with the new entry.

Set dates, units, and end items

To set the date, unit number, and/or end item of the currently active revision rule, choose **Tools→Revision Rule→Set Date/Unit/End Item**.

Teamcenter displays the **Set Date/Unit/End Item** dialog box into which you can enter a date, a unit number, and an end item for the current revision rule.

The first time you make a change to a saved revision rule (including setting a date, unit number, or end item), Teamcenter creates a modified copy of the rule and applies it to the window. This is the rule to which you make the unit number change.

If any date, unit number, or end item has been explicitly set in the current rule, you cannot change the value in this dialog box and the relevant boxes are grayed out.

> Tip:
>
> You can add a button to the toolbar to initiate the **Tools→Revision Rule→Set Date/Unit/End Item** command, by right-clicking the toolbar, choosing **Customize**, and selecting the required button.

If you specify an end item identifier that is shared by multiple objects, Teamcenter displays the **Select Unique Item** dialog box, allowing you to select the object you require.

You can create or modify a revision rule in applications other than Structure Manager, for example, in 4G Designer. Structure Manager does display such revision rules and their entries, and you can use them to configure the structure based on the displayed entries. However, a product structure in Structure Manager and a collaborative design in 4G Designer may have different data conditions. If so, the configurations in the applications are different, even if you use the same revision rule.

Delete a rule entry

1. Select an entry in the list of entries.

2. Click **Remove** and Teamcenter removes it from the rule.

Edit the entries within a rule

You can use the arrow buttons to move entries up or down within the rule.

You can use the **Group/Ungroup** command to add entries to a group or remove them from a group.

Modify the current revision rule

Users can make modifications to the window's current revision rule. This does not affect the saved revision rule, but instead, Teamcenter creates a copy of the rule, modifies it as specified, and applies it to the window. To modify the current rule, choose **Tools→Revision Rule→Modify Current**; Teamcenter displays the **Modify Current Rule** dialog box.

Use the **Modify Current Rule** dialog box to modify the current rule with the Revision Rule Editor. The editor contains a copy of the saved revision rule that is applied to the product structure window. The name of the copied rule is the name of the original rule with **(Modified)** appended. You cannot edit the **Name** box, but you can change the **Description** box.

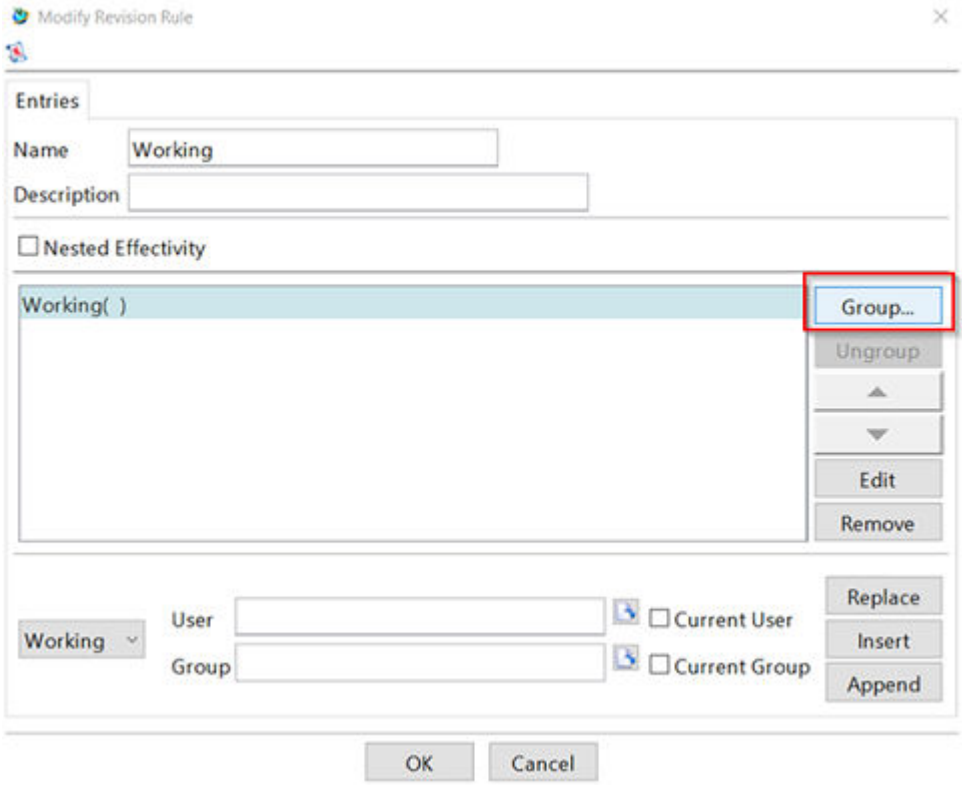
After you edit the revision rule, apply your changes by clicking **OK** or **Apply**.

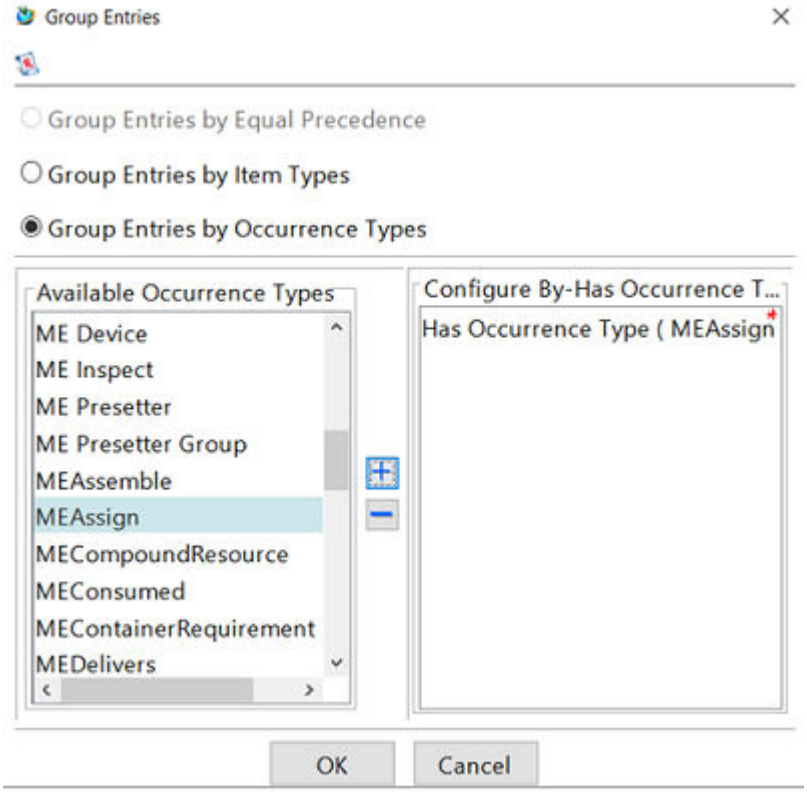
If you have write access to the original rule, you can save your changes back to the original rule by clicking **Save**.

Modify the occurrence types in existing revision rule entries

You can modify the occurrence types in existing revision rule entries grouped by the occurrence type.

1. Choose **Tools→Revision Rules→Create/Edit**.





2. In the dialog box, select one or more **Has Occurrence Type** revision rule and click the **Edit** button or double-click a single line.

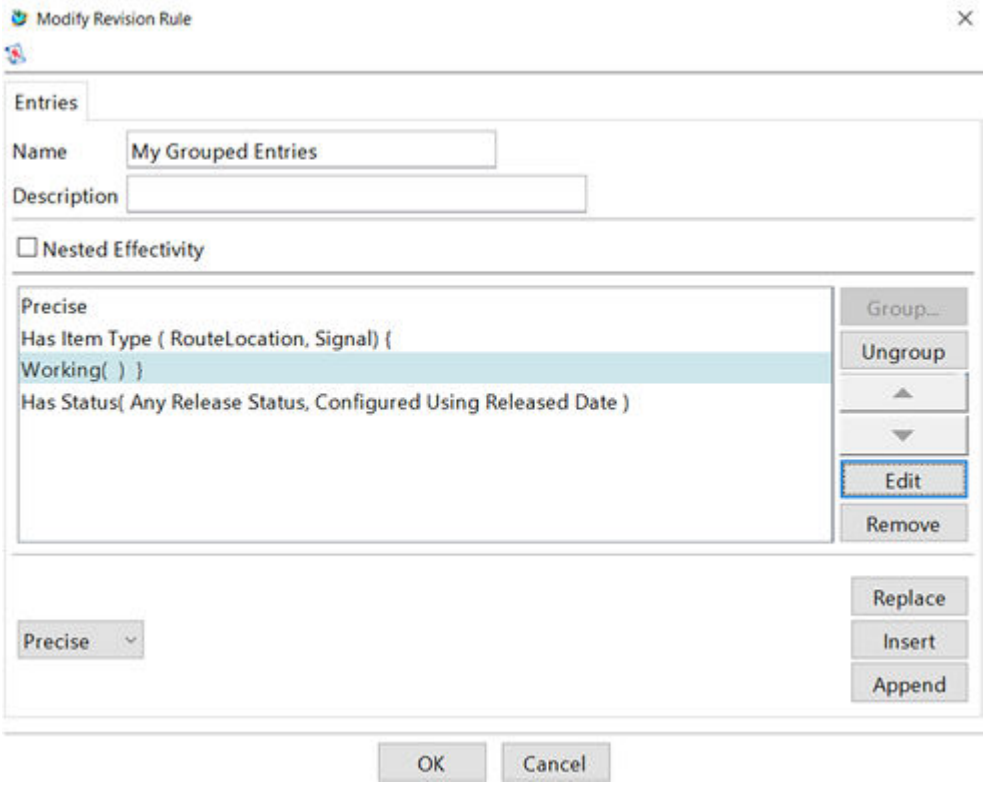
3. To add an occurrence type to the **Configure By-Has Item Type** list, select the occurrence type from the **Available Occurrence Types** list and click the **+** button.

4. To update the revision rule, click **OK**.

Modify the item types in existing revision rule entries grouped by item type

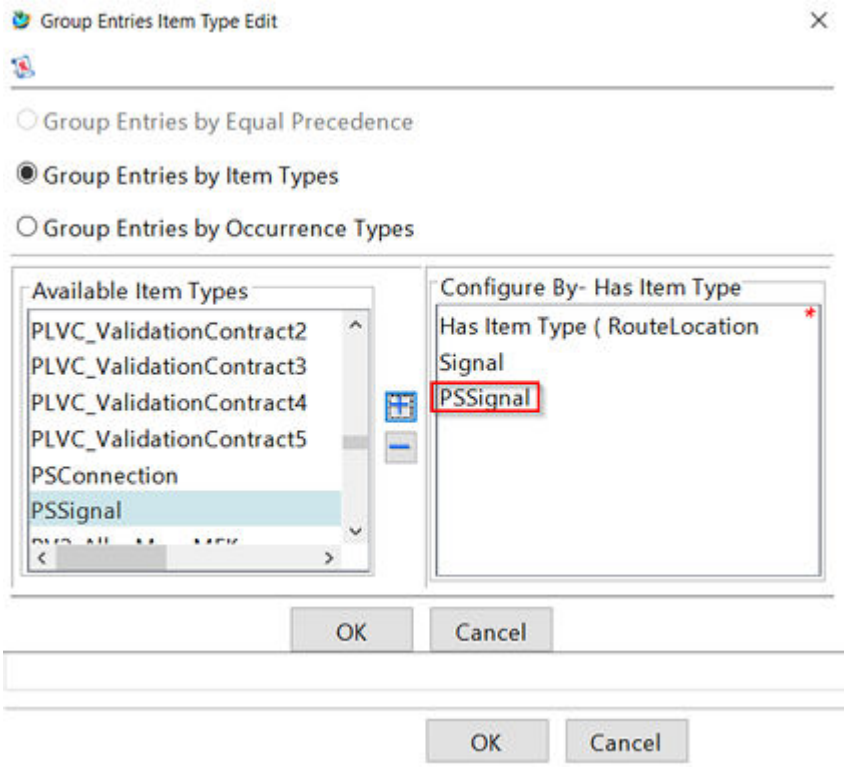
1. Choose **Tools→Revision Rules→Create/Edit**.

   Teamcenter displays the **Modify Revision Rules** dialog box, as shown in the following example.



2. In the dialog box, select one or more **Has Item Type** revision rule entries and click the **Edit** button or double-click a single line.

   Teamcenter displays the **Group Entries Item Type Edit** dialog box, similar to the following example.



3. The item types that are currently grouped in the selected revision rule entry are displayed in the **Configure By-Has Item Type** list. To add a selected item type from the **Available Item Types** list, click the **+** button. To remove a item type, select it in the **Configure By-Has Item Type** list and click the **–** button. When the list of item types is shown correctly, click **OK** to update the revision rule.

> Note:
>
> You must check at least one item type; otherwise, Teamcenter displays an error message.

Create a revision rule with an override clause

You can create a revision rule with an override clause that engineers can use this revision rule for various tasks, such as marking the structure as precise, checking temporary changes, performing what-if analyses, and analyzing changes to the structures based on business requirements.

Procedure

1. Create a folder that can be used by the users as an override folder.

To create a folder	Do this
In Active Workspace	a. In the **EXPLORER** tile, navigate to a location where you want to create a folder and click **More Commands** ▫▫▫ > **New** ✳ > **Add** ⊕.

To create a folder	Do this
	b. On the **Add** panel, select **Folder** type. c. Enter a name and description for the folder. d. Click **Add**.
In the rich client	a. In My Teamcenter, click **File→New →Folder**. b. In the **New Folder** dialog box, select **Folder** and click **Next**. c. Enter a name and description for the folder. d. Click **Finish** and **Close** to close the New Folder dialog box.

2. In Active Workspace, set the **AWC_display_configured_revs_for_pwa** preference to **false**.

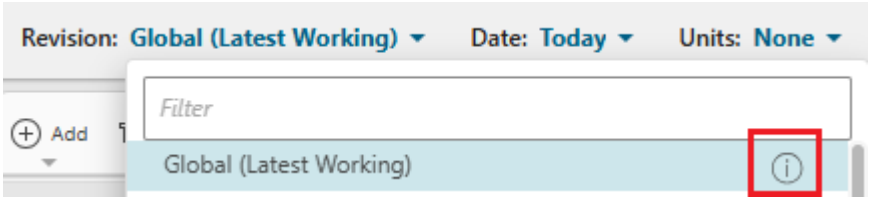
   Setting the **AWC_display_configured_revs_for_pwa** preference to **false** allows an engineer to copy the revision of an item, instead of copying the item itself, to the folder.

3. Create a revision rule with an override clause.

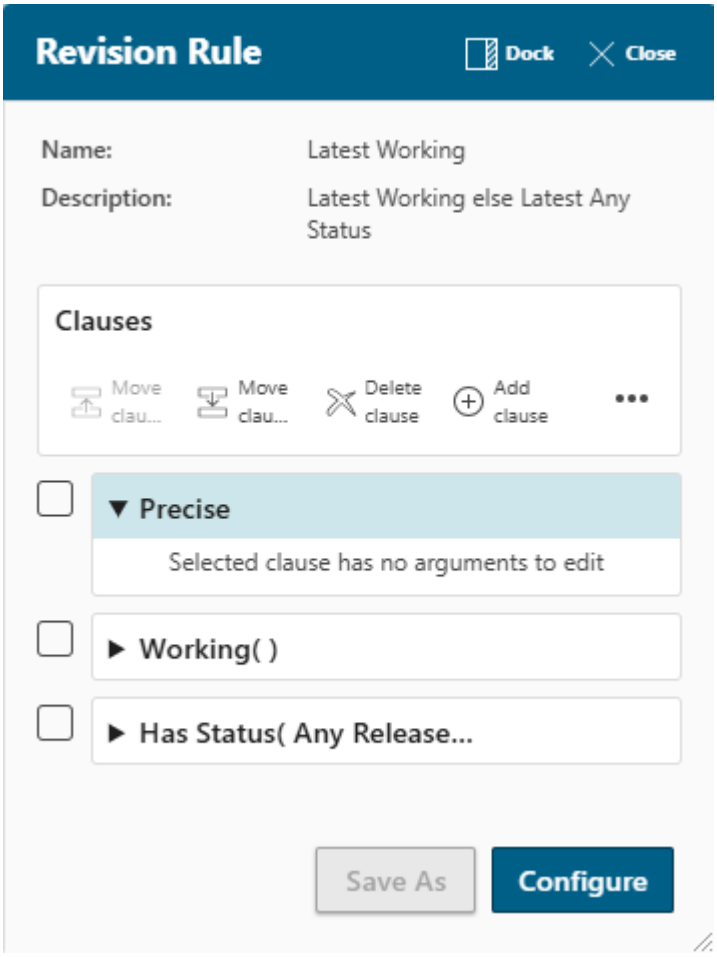
   - In Active Workspace, perform the following steps:

     a. Open a structure and click the **Revision** list.

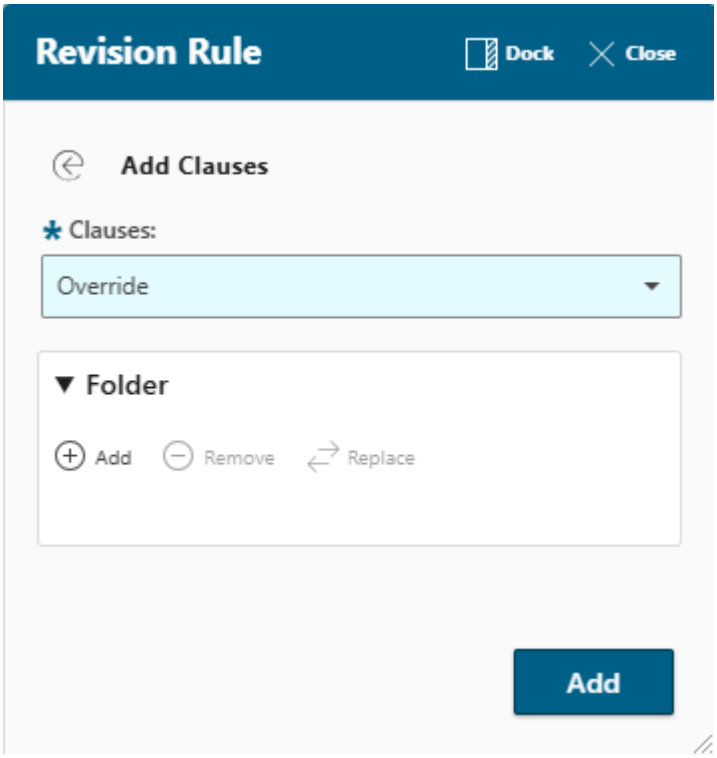
     b. Click ⓘ next to the revision rule that you want to modify.



     c. In the **Revision Rule** panel, click ⊕ **Add clause**.

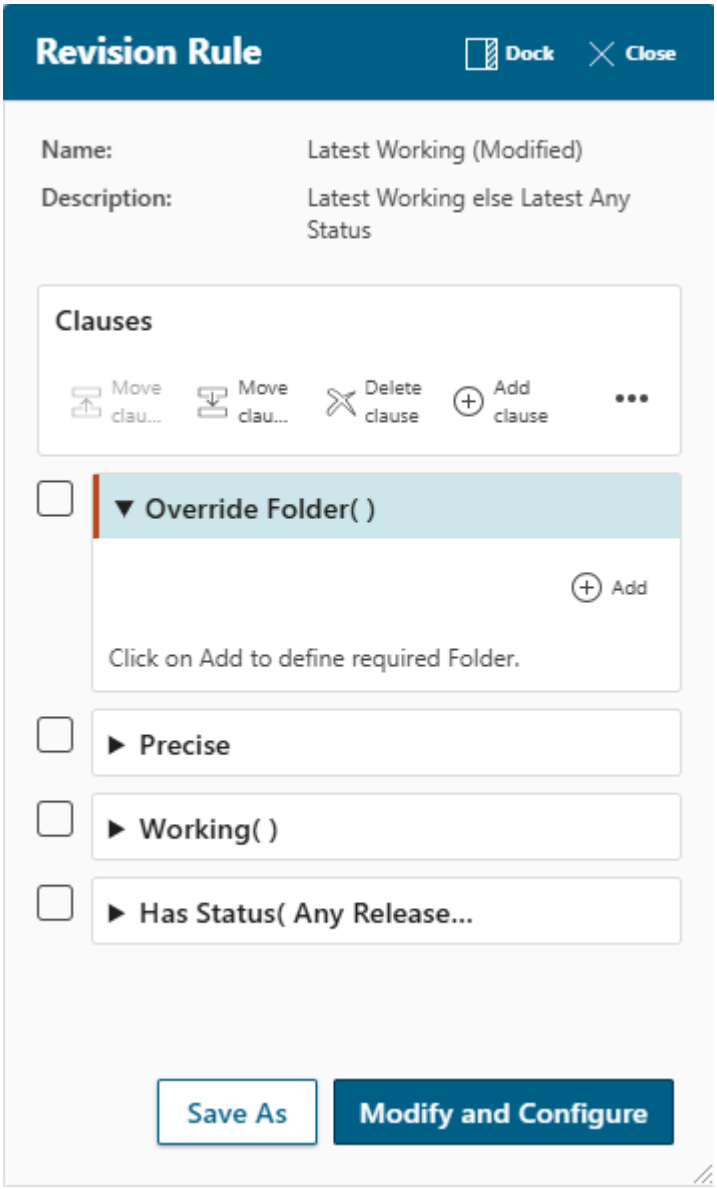


d. On the **Clauses** list, select **Override** and click **Add**.



e. Modify the clauses and set the precedence of the **Override** clause considering your requirement.

Considering your requirement, you can add multiple clauses with override folders to the revision rule.

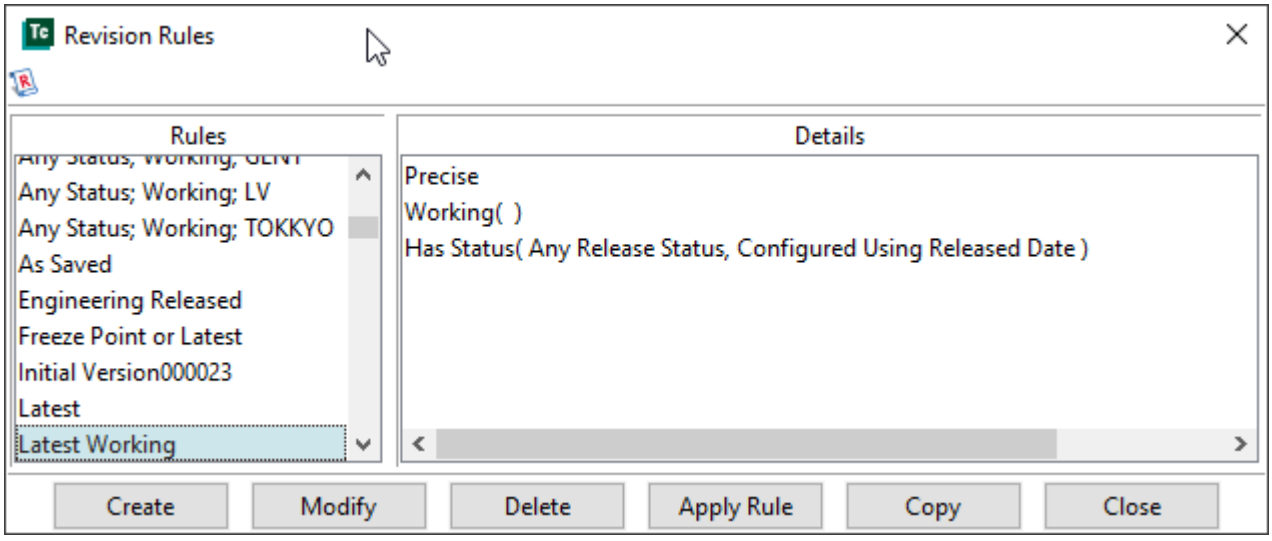


f. Click **Modify and Configure** to modify the selected revision rule.

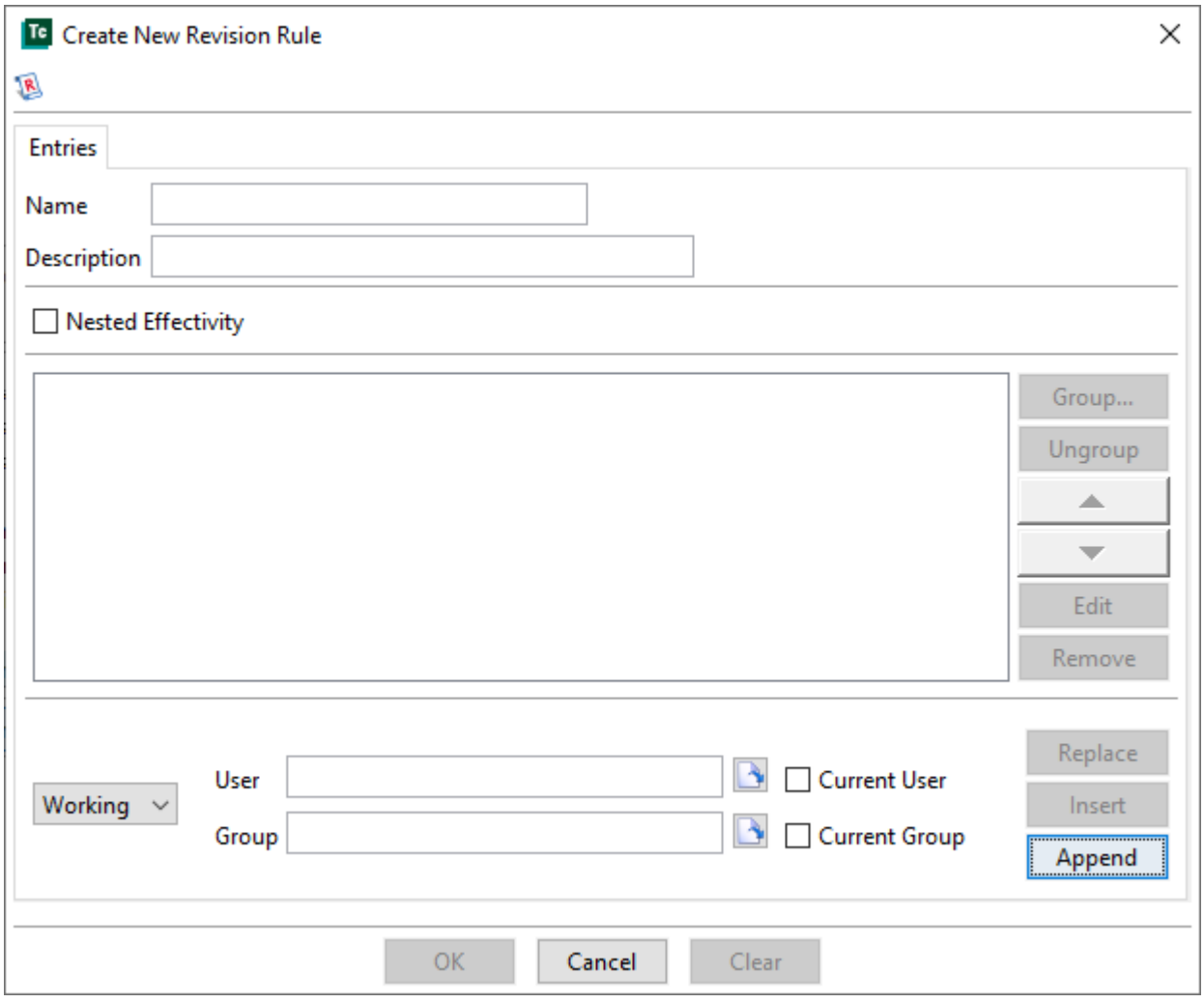
   Alternatively, click **Save As** and then enter a name and description for saving this revision rule with a new name.

- In the rich client, perform the following steps:

  a. In the Structure Manager application, click **Tools→Revision Rule→Create/Edit**.

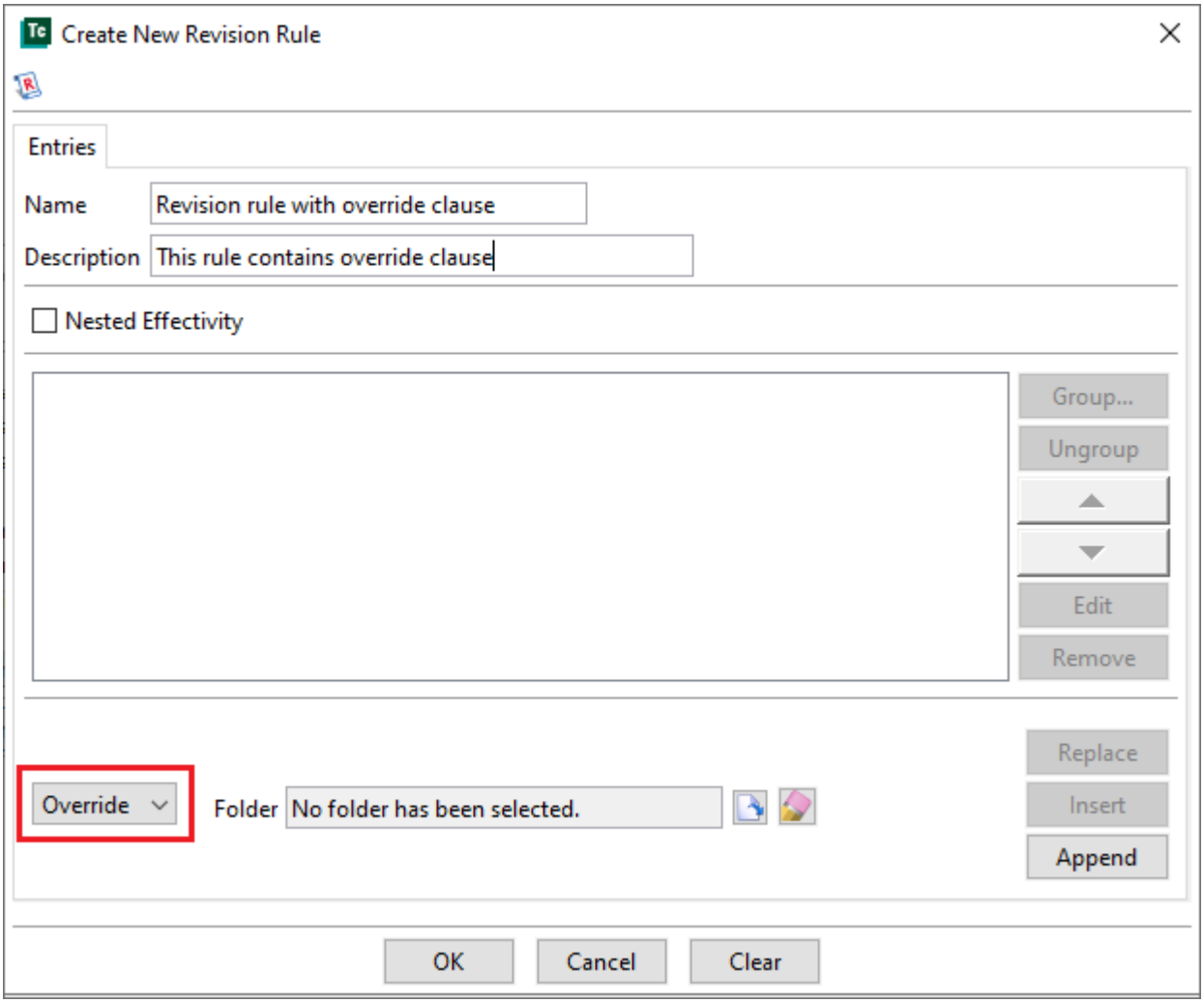


b. In the **Revision Rules** dialog box, select a rule that you want to use as a basis for creating the new revision rule in the **Rules** section, and click **Create**.



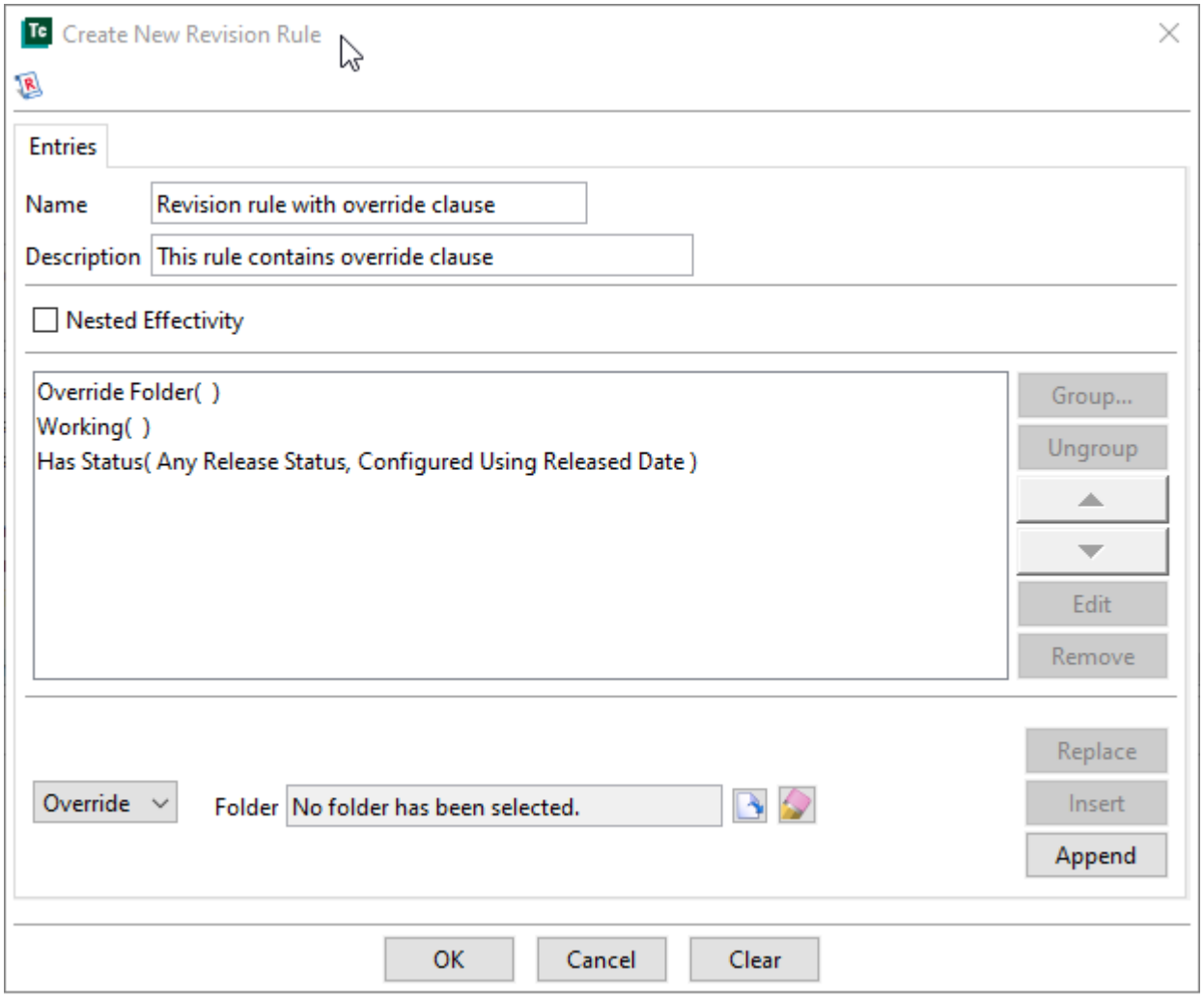
c. In the **Create New Revision Rule** dialog box, enter a name and description for the revision rule.

d. Select **Override**, and choose **the folder that you created in the previous step**.



e. Modify the clauses and set a precedence for the **Override** clause.

Considering your requirement, you can add multiple clauses with override folders to the revision rule.



f. Click **OK**.

The revision rule is created.

Set the default revision rule for a product structure

Teamcenter provides default revision rules that may be used at your site. **TC_Config_Rule_Name** specifies the default revision rule for structures. Set the **AWBUseDefaultRevisionRule** preference to `True` so that the configuration panel uses the existing **TC_Config_Rule_Name** as the default revision rule.

> Note:
>
> The **PSEShowUnconfigdEffPref** preference determines whether structure lines with effectivity that do not configure for the current revision rule or incremental changes are shown by default in a new Structure Manager window. If a structure line is shown even though its effectivity evaluates

to **false** for the current revision rule or incremental changes, you can use the following properties to identify the lines that do not configure for the current variant rule:

```
bl_has_date_effectivity=Y
bl_is_occ_configured=
```

Change the default revision rule for a workset

By default, when users open a workset, it uses the **Precise Only** revision rule. This is because the default value of the **TC_workset_config_rule_name** preference is **Precise Only**. You can change the default revision rule to another revision rule based on your site requirement. However, the new default revision rule must contain **Precise** as its first clause.

Provide access privilege for revision rules

Evaluate revision rules for read access

By default, Teamcenter applies a revision rule to configure a product structure before it applies an access rule. As a result, if a higher revision rule entry configures a particular revision of an item and the user has no read access to this revision, the structure line in Structure Manager shows an **UNREADABLE** stub entry.

You can optionally configure Teamcenter to continue evaluation when expanding the structure against the revision rule until it encounters a revision of the item to which the user has read access. However, if it finds no accessible item revision for any of the entries, it shows an **UNREADABLE** stub, as before. To allow Teamcenter to continue revision rule evaluation by making read access checks in this way, set the **PSE_check_read_access_for_Rev_Rule_entry** preference to **TRUE**. Structure Manager shows the next item revision to which you have access. If this preference is set to **FALSE**, evaluation stops when the first inaccessible configured item revision is encountered.

Unreadable lines may be shown or hidden, depending on the setting of the **BOM_hide_unreadable_lines** site preference. If the preference is set to **None**, unreadable lines are shown and labeled as **UNREADABLE**; this is the default setting. If the preference is set to **All**, unreadable lines are hidden. If the preference is set to a group name, unreadable lines are hidden from users in the specified group.

- If the user has no access to an item or GDE (generic design element) in an imprecise structure, the line is not displayed.
- If the user has access to the item in a precise structure but not the configured item revision, the line is displayed.
- If the BVR is precise and the user has no access to the item revision, the line is not displayed.
- If the BOM has only unreadable lines, the parent is shown as a leaf node.

Unreadable lines do not participate in roll ups, configuration, BOM compare, accountability checks, duplication, import/export, and searches. Hidden lines cannot be used as global alternates or substitutes.

You can use any of the following revision rule entries to show configured item revisions to which the user has read access:

- **Latest Working**
- **Status**
- **Latest Revision**
- **Override**

You can include more than one of these entries in the same revision rule to find accessible item revisions.

This behavior applies only to imprecise structures and does not affect the evaluation of precise structures. Similarly, there is no effect if you configure a structure by unit number or date effectivity.

Configure privileged and unprivileged users

User privileges determine if a user can create and modify revision rules, or only apply rules created by others. Specifically, privileged users have access to all the revision rule menu commands, while unprivileged users have access only to a subset of the commands. Consequently, unprivileged users cannot create or delete revision rules, and have only limited ability to modify revision rules without saving the changes.

Your Teamcenter administrator can restrict user access to these menu commands. Use the Command Suppression application to determine those roles that are privileged and so restrict access to the **Revision Rules** menu commands.

Control access to revision rules

Use Access Manager to control user access to revision rules. You can limit read access to control the users who can see and use a revision rule. You can use this technique to reduce the number of inapplicable revision rules that are presented to ordinary users, or to restrict rules to certain groups of users. You can use write access to control the users who can modify a revision rule.

You can apply an access rule globally to all rules using a class revision rule or other attribute, (for example, **OwningGroup**) if you created the revision rules appropriately. You can add object ACLs to specific revision rules for exception cases. A typical default access rule and rule tree ACL follow:

Access rule:

```
HasClass (RevisionRule) -> Private Rev Rule ACL
OwningGroup (dba) -> Public Rev Rule
```

Private revision rule ACL:

This ACL prevents Teamcenter displaying privately created revision rules to all users. Only the owning user and system administrator have access to the private rule. You can define an entry for **Owning User** that gives access to all users in the owning group. Alternatively, you could add it as an object ACL to the specific rule.

```
Owning User: Read, Write, Delete, Copy, Change
System Administrator: Read, Write, Delete, Change
World: No Read, No Write, No Delete, No Copy, No Change
```

Public revision rule ACL:

This ACL ensures that public revision rules are visible to all users. It also only allows users with a **configuration** role or members of a system administration group to modify public rules. You should control these permissions carefully, as unintended modification of revision rules can have significant consequences.

```
Role = Configurator: Write
System Administrator: Write, Delete, Change
World: Read, No Write, No Delete, No Copy, No Change
```

Improve the performance of BOM expansion during revision configuration

You can improve the performance of BOM expansion during revision configuration by using the **Teamcenter Revision Configuration Accelerator Service** service. Currently, this service supports the **Working** clause and the **Has Status** clause in a revision rule.

You must install the **Teamcenter Revision Configuration Accelerator Service** service and make sure that it is running. To install the service:

- In TEM, go to **Server Enhancements** > **Database Daemons** > **Teamcenter Revision Configuration Accelerator Service**.

  OR

- In Deployment Center, go to the **Components** tab and add the **Database Daemon** component. Under **Database Daemon Services**, select the **Teamcenter Revision Configuration Accelerator Service** check box.

The service runs in the background at regular intervals and updates the revision configuration results for the modified revision data.

18. Configure structure effectivity

Migrate legacy effectivity data to a new effectivity object

Earlier, the legacy effectivity data was stored in the **CFM_date_info** object. As this object is now obsolete, you must migrate the effectivity data to the new effectivity object called **Effectivity**.

To do this, in the Teamcenter command prompt, run the **effupgrade** utility in the **CFM date info occurrence effectivity upgrade** mode as follows:

```
effupgrade -cfm_date_info_occ
```

Along with the **-cfm_date_info_occ** mode, you must use the **Report**, **Upgrade**, and **Cleanup** submodes. You must use the **Report** submode first, then the **Upgrade** submode, and finally, the **Cleanup** submode. Optionally, you can use the **Upgrade** and **Cleanup** submodes together. As a system administrator or a business user, you must not modify any effectivity data while the utility is running in these submodes. If any data is modified, you must run the utility again, starting with the **Report** submode.

The **Report** submode creates a text report file for the legacy CFM date info data. The report file is used later by the **Upgrade** and **Cleanup** submodes. You must not modify this file. It is saved in a temporary folder. After this submode is run completely, the console displays the name of the text report file along with its file path. The following is the syntax for this submode:

```
effupgrade -cfm_date_info_occ -report
```

The **Upgrade** submode migrates the CFM date info data collected in the report file to the **Effectivity** object. After this submode is run completely, the migration status is updated to the report file. This same report file is then used by the **Cleanup** submode. The following is the syntax for this submode:

```
effupgrade -cfm_date_info_occ -upgrade -reportFile <report file name with
path>
```

The **Cleanup** submode cleans up the legacy **CFM_date_info** object after its data is migrated. If you provide the report file as an input, the cleanup is performed using the report file. After this submode is run completely, the cleanup status is updated to the report file. However, if you do not provide the report file, all the orphan **CFM_date_info** objects are cleaned up from the database. The following is the syntax for this submode:

```
effupgrade -cfm_date_info_occ -cleanup -reportFile <report file name with
path>
```

For more information about the **CFM date info occurrence effectivity upgrade** mode, in the Teamcenter command prompt, type the following command:

```
effupgrade -occ_eff_help
```

To view all the information available on the **effupgrade** utility, type the following command:

```
effupgrade -h
```

Display date effectivity without end item

If an end item is associated with a date effectivity, the end item is displayed along with the date effectivity. However, if an end item is not available, the text *(NONE)* is displayed after the date effectivity. To set how the date effectivity is displayed, set the following preference.

Preference	Value	Description
PSE_show_eff_empty_end_item	**true** This is the default value.	The date effectivity is displayed with the date followed by the end item (if an end item is available). For example, **2-Nov-2023 14:30 to 29-Apr-2023 14:30 to UP (037083)**
	false	Only date is displayed. For example, **3-Nov-2023 14:30 to UP (NONE)**

Configure nested effectivity

To configure nested effectivity, your Teamcenter administrator must set the **CFMEffConfigItemProperties** preference. This preference contains the item properties that identify configuration items. For example, the following entry defines the name of the type of item as the identifying attribute:

isConfigurationItem
object_type:object_name

Each entry in the list must be followed by the corresponding values. For example, the following example states that a configuration item is of the **Product** or **Vehicle** item type:

EffectivityConfigurationItemProperty.isConfigurationItem=true
EffectivityConfigurationItemProperty.object_type.object_name=
Product
Vehicle

Create a configuration item for the rich client

You do not create a configuration item separately, but you can identify an item as a configuration item when you create it. To do this, ensure the **Configuration Item** check box is selected in the **New Item** wizard before you click **Finish**. The **Configuration Item** check box appears in the **New Item** wizard only if it is enabled using the Business Modeler IDE.

Alternatively, you can convert an existing item to a configuration by changing the value of its **Configuration Item?** property from **False** to **True**. You can edit the item properties by right-clicking the item and choosing **Edit Properties** and clicking the **All** link.

Note:

The Teamcenter administrator can perform the following steps to add the **Configuration Item** check box to the **New Item** wizard.

1. In the Business Modeler IDE, open the item business object type.
2. Click the **Operation Descriptor** tab and select the **CreateInput** tab.
3. Select the **is_configuration_item** property.
4. On the **Property Constants** tab, select the **Visible** property constant.

Create a revision rule for nested effectivity

When you create a revision rule for a nested effectivity scheme, ensure you select the **Nested Effectivity** check box. This action adds a **Nested Effectivity** entry to the revision rule and ensures that Teamcenter modifies the revision rule according to the mapping on any configuration items it encounters. If you do not check this check box, Teamcenter ignores any configuration items. The position of the **Nested Effectivity** entry in the rule is not significant and never changes.

The following example structure has date effectivity and an end item defined on the item revision of each part:

Item	Status, effectivity	Note
NE-A1000/A (CI) – Product X		
NE-A2000/A (CI)	Released, EI: NE-A1000, 15 July-UP	Rev Effect wrt NE-A1000 – Product X
NE-A100/A	Released, EI: NE-A2000, 20 July-UP	
NE-A200/A	Released, EI: NE-A2000, 20 July-UP	

Item	Status, effectivity	Note
NE-PP30/A	Released, EI: NE-A2000, 20 July-UP	
NE-PP10/A	Released, EI: NE-A2000, 20 July-UP	
NE-PP20/A	Released, EI: NE-A2000, 20 July-UP	
NE-A3100/A (CI)	Released, EI: NE-A1000, 16 July-UP	Rev Effect wrt NE-A1000 – Product X
NE-A400/A	Released, EI: NE-A3100, 23 July-UP	
NE-PP70/A	Released, EI: NE-A3100, 25 July-UP	
NE-PP60/A	Released, EI: NE-A3100, 25 July-UP	
NE-A5000/A (CI) – Product Y		
NE-A2000/A	Released, EI: NE-A5000, 15 July-UP	Rev Effect wrt NE-A5000 – Product Y

The resulting configuration item effectivity mappings are:

- Mapping on NE-A2000 (CI):

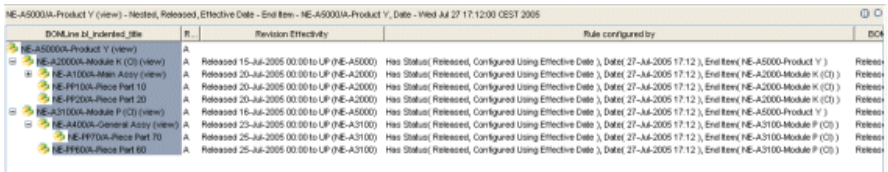
  NE-A1000 (CI) – Product X : NE-A2000 (CI)
  NE-A5000 (CI) – Product Y : NE-A2000 (CI)

- Mapping on NE-A3100 (CI):

  NE-A2000 (CI) – Product Y : NE-A3100 (CI)

If you apply the following revision rule, the structure is configured as shown in the following figure:

Status = Released, Configured by Effective Date
Working
Nested Effectivity
EI = NE-A5000
Date = 27 July 2006



Nested effectivity 1

Conversely, if you apply the following revision rule with no nested effectivity entry, the structure is configured as shown in the following figure:

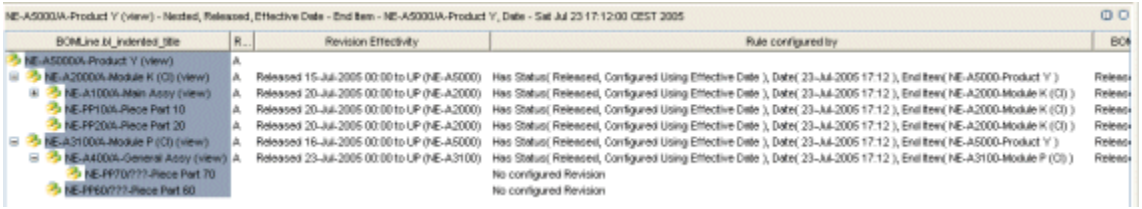
Status = Released, Configured by Effective Date
Working
EI = NE-A5000
Date = 16 July 2005

Nested effectivity 2

Teamcenter ignores the mappings, so the end item does not switch from A5000 to NE-A2000/NE-A3100 at the configuration items. Consequently, none of the components below the configuration items are configured, as the end item on the effectivity is not NE-A5000.

If you apply the following revision rule, Teamcenter applies effectivity mapping due to the **Nested Effectivity** entry and the structure is configured as shown in the following figure. However, some of the components are not effective on 23 July and have no configured revision; Teamcenter marks these components as **???**.

Status = Released, Configured by Effective Date
Working
Nested Effectivity
EI = NE-A5000
Date = 23 July 2005



Nested effectivity 3

Allow users to view shared effectivity information

Users can view the shared effectivity associated to a structure in the **Table** view. The effectivity is shown when the **Table** view is selected in the **Overview** tab. To view effectivity in the **Table** view, add the following string in the **Awb0ContentTableItemRevUiConfigCots.xml** file:

```
<ColumnDef objectType="Awb0ConditionalElement"
propertyName="awb0ElementEffId" width="4000"/>
```

Enable multi-unit effectivities

Teamcenter allows you to configure product structure occurrences of an assembly based on specified multiple end items and the unit effectivity ranges for each of those end items. You can do impact analysis and eliminate the duplicate work required to maintain different product structures and complicated manual reconciliation.

A combination of multiple end items and range of units for each end item used to configure product structure occurrences is referred to as a *multi-unit configuration*. To enable the creation of multi-unit effectivities, the administrator must set the **Fnd0EnableMultiUnitConfiguration** global constant to **true** at each site with the Business Modeler IDE (BMIDE).

1. Open BMIDE with the foundation template.

2. Search for the **Fnd0EnableMultiUnitConfiguration** global constant and change the value to **true**. By default, it is set to **false**.

3. Create a custom template with the changes in BMIDE.

   For more information, see *Creating, deploying, and packaging templates* in *Configuring Your Business Data Model in BMIDE* on Support Center.

4. Deploy the template with the changes.

   For more information, see *Deploying templates* in *Configuring Your Business Data Model in BMIDE* on Support Center.

Control users who can set effectivity

To control the users who may set effectivity, add the following access rules at an appropriate place at the top level of the rule tree:

- **Has Type (Release Status)→Create Effectivity Users**
- **Has Type (Effectivity) - Edit Effectivity Users**

The release status rule controls who has write access to the release status and consequently can attach effectivity objects to it. This also determines who can initially create effectivity. Similarly, the effectivity rule controls who can edit an existing effectivity object.

19. Administer solution variants

Set up workflows to create or update solution variants

To allow users to create solution variants using a workflow, you must create a workflow process template by using the **SMC0-create-solution-variants** handler.

To allow users to update solution variants using a workflow, you must create a workflow process template by using the **SMC0-update-solution-variants** handler.

Set up the column configuration for solution variants

You must perform the following steps so that the columns specific to solution variants are displayed on Active Workspace:

1. Verify if the **Active Content Structure**, **Product Configurator Support for Active Content Structure** (**smc1activeworkspacebom**), and **Solution Variant Support for Active Workspace** are installed.

2. Verify if the **smc1activeworkspacebom** template is set up in BMIDE.

3. (Optional) Add the custom properties that you want to be displayed as columns specific to solution variants on Active Workspace in the *Smc1SVPreviewUiConfigCots.xml* file. Additionally, add the custom properties that you want to be included in the **Solution Variants** table present on the **Overview** tab or in the **Solution Variants** section present on the **Solution Variants** tab of a configurable assembly (variable part) in the *SolutionVariantsProvider.xml* file. These files are located in the *TC_ROOT\smc1activeworkspacebom\data* folder.

   Use the **import_uiconfig** utility to import these files so that the custom properties are displayed on Active Workspace.

For detailed information on importing and merging column configurations, see the *Active Workspace Customization* help on Support Center.

Specify the source name and ID format for creating solution variants

In Teamcenter, if the **Enable Multi-Level Creation** option is enabled, you can use the **PSESolutionVariantItemProperties** preference to specify a format for the source name and the ID when creating a solution variant. This preference accepts a combination of values, which can be specified manually or can be system generated.

By default, this preference does not contain any values. If you do not specify any value or specify an invalid value, Teamcenter uses the name from the source structure and automatically generates the ID.

Procedure

1. Specify the internal names of the properties from the source Item Revision, separated by the plus (**+**) character. No spaces are allowed.

   - To define the ID, use the following format: item_id_format=<itemrev_prop1>[+<itemrev_prop2>]*. For example, "item_id_format=custom_prop1+item_id+custom_prop2".

   - To define the solution variant object name, use the following format: object_name_format=<itemrev_prop1>[+<itemrev_prop2>]*. For example, "object_name_format=custom_prop1+object_name+ custom_prop2".

   The resulting object name or ID is a combination of the specified property values and the system generated ID if the specified property is **item_id**, separated by the minus (**-**) character, for example, "000260-custom_prop1_value". The source name or ID gets truncated if it exceeds the limit for the maximum allowed length.

2. Save your changes.

> Note:
>
> Teamcenter continues to support the earlier format for specifying values, for example, **item_id=bl_item_object_desc** and **object_name=bl_rev_object_desc**.

Define the reuse of a solution variant

By default, before creating a solution variant, Teamcenter compares the content of the configured source structure with solution variants that are already available. The content of the structure is based on the applied variant configuration. While comparing, if a matching solution variant is found, it is reused instead of creating a new one.

Teamcenter does the content comparison because the **SolutionVariantReuseBasedOnStructureMatch** preference is set to **true**, by default. For each unique saved variant rule (SVR) applied to the source configurable structure, if a matching solution variant is found, it is reused.

If you want new solution variants to be created for each each unique SVR, you must set this preference to **false**. On doing so, Teamcenter does not compare the content, and always creates new solution variants. However, for lower-level *reuse* assemblies where a subset variability scope is applied, solution variants are reused if the resulting features of the subset SVR matches an existing solution variant. A subset variability scope is a scope that belongs to the variability scope applied at the topmost level of the structure.

Disallow updating an existing solution variant

You can prevent updating an existing solution variant of type *Reuse* by configuring the statuses, so that a business user can obsolete it and create and then use a new solution variant.

To do that, you use the **SolutionVariantDisableUpdateStatusList** preference. You can add all valid Teamcenter statuses as values to this preference. The default value available in this preference is **Obsolete**. You can add more statuses to the list of statuses as per your business requirement.

When the user sets any of the statuses in the preference on a solution variant, automatic updates to the solution variant are disabled. The business user can replace this solution variant by either creating a new solution variant for its 150% configurable structure or by updating its parent solution variant.

Configure how a user creates, views, and updates solution variants

You can configure how a business user creates, views, and updates solution variants by using the **SolutionVariantProviderAvailable** preference.

Preference value	Solution variants table location	User action
False (default value)	**Solution Variants** tab > **Solution Variants** section	A user sets the **Solution Variant Category** to **Reuse** and updates a solution variant manually.
True	**Overview** tab > **Solution Variants** section	A user updates a solution variant using a workflow from the same location.

Specify the occurrence properties to be synced from the source structure to the solution variant while the solution variant is updated

You can specify the occurrence properties that are to be synced from the source structure to the solution variant while the solution variant is being updated. To do that, you use the **SolutionVariantSyncProperties** preference.

By default, this preference contains the following values:

- **bl_quantity**
- **bl_plmxml_occ_xform**
- **bl_occ_effectivity**
- **bl_ref_designator**

- **bl_sequence_no**
- **AIE_OCC_NAME**
- **AIE_OCC_ID**
- **AIE_Exported**

You can modify the preceding list to add any additional occurrence properties, such as Occurrence Note or any other custom occurrence properties. A property value must be an internal name of a BOM line property. The substitutes for a structure element are synced automatically. You need not include those as values in this preference.

In addition to carrying over the absolute data properties, if you want to carry over the **Position Designator**, **ID in Context**, and **Usage Address** properties, add the following values to the **SolutionVariantSyncProperties** preference:

- **bl_abs_occ_id**
- **bl_position_designator**
- **bl_usage_address**

20. Configure Product Configurator variants

Modify AND and OR operators in variant conditions

1. In your Teamcenter installation directory, search for the **tc_text_locale.xml** file, for example, *TC_ROOT***lang\textserver\en_US\tc_text_locale.xml**.

2. Create a backup of this file.

3. Open the file and search for the **k_variant_op_and** and **k_variant_op_or** values and modify them.

   Example:

   ```
   <key id="k_variant_op_and">AND</key>
   <key id="k_variant_op_or">OR</key>
   ```

   Change as follows:

   ```
   <key id="k_variant_op_and">&amp;</key>
   <key id="k_variant_op_or">|</key>
   ```

4. Start Teamcenter and confirm that formula representation is now changed from **AND, OR** to **&, |** successfully.

Enable the tracking of variant changes

To enable the tracking of variant changes, set the **CM_track_variant_condition_changes** preference to **ChangeNotice** types.

When users save changes to a structure containing variants, if this preference is **ON** for the current change type, Teamcenter tracks variant changes between the solution item and the impacted item.

21. Configure classic variants

Replace variant condition column with variant formula column for classic variants

The **Variant Conditions** column is available by default in Structure Manager. This column is applicable only to classic variants. If a structure has Product Configurator variability, this column does not display any values. In such cases, you can replace it with the **Variant Formula** column for all users.

To know how to add or remove columns, see Insert or remove a property column.

Modifying the sequence of columns in Structure Manager

- If an individual user modifies the sequence of columns, the system creates user level instances of the **PortalPSEColumnsShownPref**, **PSEDisplayNameWithTypeColumnsShownPref** and **PortalPSEColumeWidthsPref** preferences and rearranges the columns for that user.

  If users do *not* modify the sequence of columns, these preferences remain as site level preferences.

- If there are no user level instances of the **PortalPSEColumnsShownPref**, **PSEDisplayNameWithTypeColumnsShownPref** and **PortalPSEColumeWidthsPref** preferences, the system administrator modifies site level preference values by changing the **bl_variant_condition** value to **bl_formula** value in the *TC_DATA***tc_preferences.xml** file.

- If there are any user level instances of the **PortalPSEColumnsShownPref**, **PSEDisplayNameWithTypeColumnsShownPref** and **PortalPSEColumeWidthsPref** preferences, a system administrator can run the **preferences_manager** utility in cleanup mode to modify the preference values from **bl_variant_condition** to **bl_formula** for a few or all users.

Running the preferences_manager utility

As a systems administrator, you can run the **preferences_manager** utility in cleanup mode to modify the preference values for specific users or all users.

While running the utility, if you specify the **-u_target** argument with a comma separated list of users who have user instances of preferences, the columns are changed only for those users. If you do not specify this argument, columns are modified for all users who have user instances of preferences.

Run the utility in dry run mode:

preferences_manager -mode=*cleanup* **-u=***admin_user* **-p=***password* **-g=***dba* **-action_file=***actionfile.xml* **-u_target=***user1,user2* **-dry_run**

actionfile.xml file contents:

```
<?xml version="1.0" encoding="us-ascii"?>
<actions>
    <action name="remove">
    </action>

    <action name="replace">
        <preference name="PortalPSEColumnsShownPref">
            <value to_replace="bl_variant_condition">bl_formula</value>
        </preference>
        <preference name="PSEDisplayNameWithTypeColumnsShownPref">
            <value to_replace="BOMLine.bl_variant_condition">
                 BOMLine.bl_formula</value>
        </preference>
    </action>

    <action name="add">
    </action>

    <action name="rename">
    </action>
</actions>
```

On running the utility in dry run mode, the log file specifies for which all users the preferences would be modified in non dry run mode.

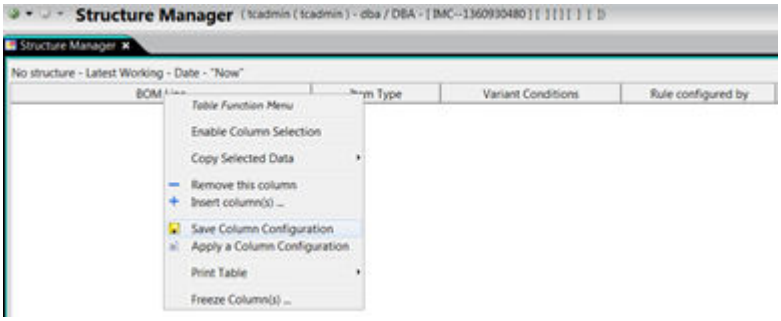
Run the utility in non dry run mode:

preferences_manager -mode=*cleanup* **-u=***admin_user* **-p=***password* **-g=***dba* **-action_file=***actionfile.xml* **-u_target=***user1,user2*

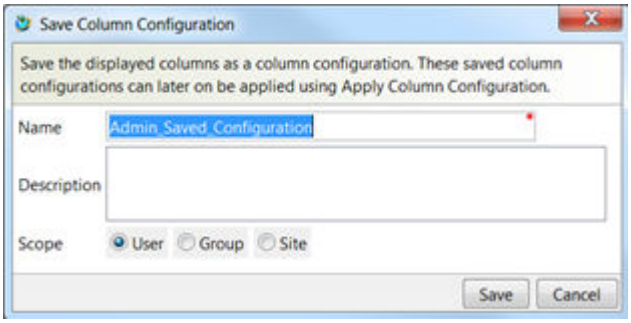
Saving column configuration

The following is an example of how a user saves a column configuration and the system administrator runs the **preferences_manager** utility to change all such column configuration preferences created by users.

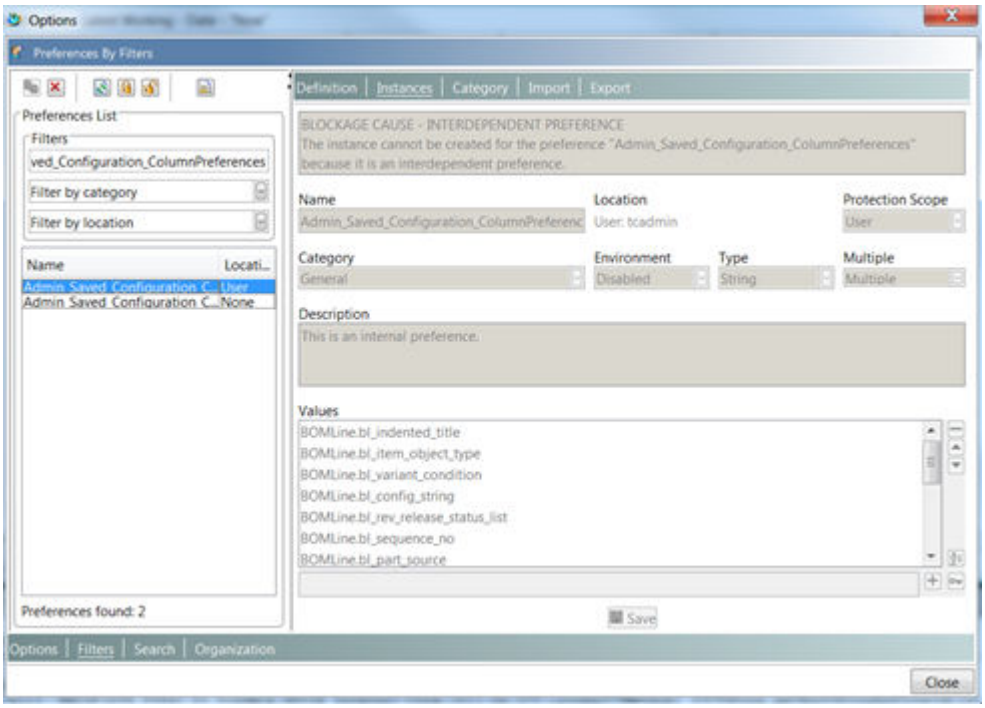
1. In Structure Manager, right-click and choose **Save Column Configuration**.



2. Save the column configuration.

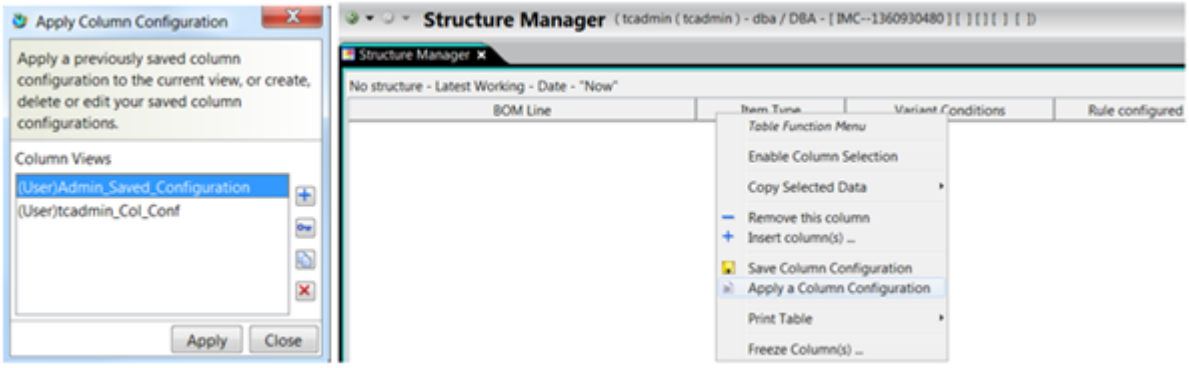


The system creates new preferences.

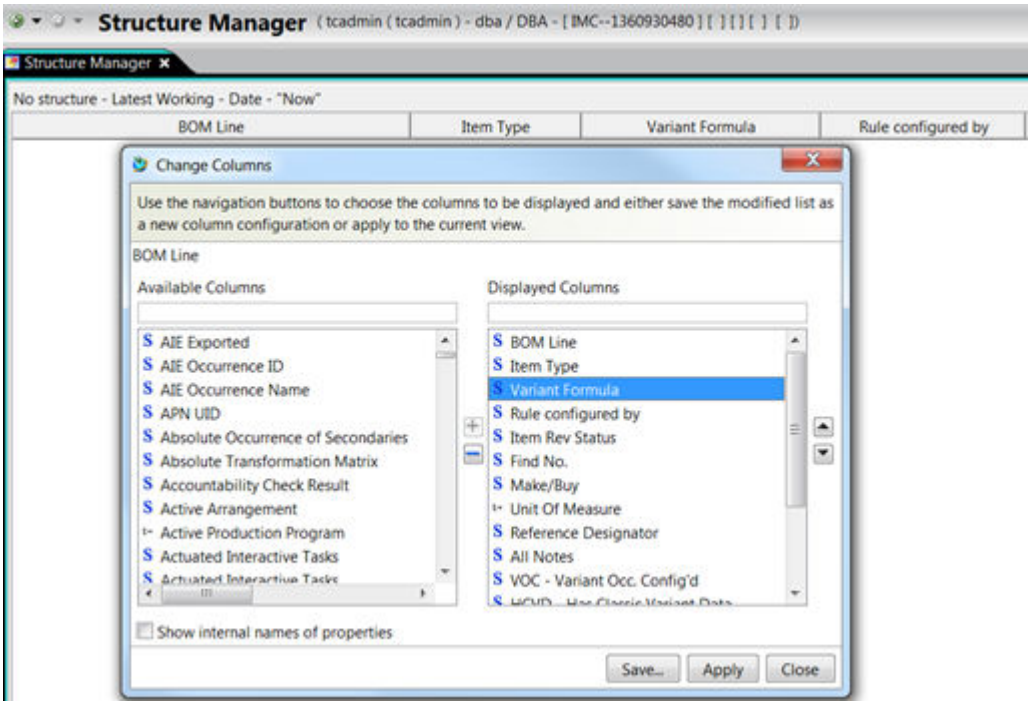


3. Apply the column configuration. To do so, right-click and choose **Apply a Column Configuration.**

> Note:
>
> You must apply the column configuration only after the system administrator runs the **preferences_manager** utility.



The **Variant Formula** column is displayed after applying the column configuration.



Modify AND and OR operators in variant conditions

1. In your Teamcenter installation directory, search for the **tc_text_locale.xml** file, for example, *TC_ROOT***lang\textserver\en_US\tc_text_locale.xml**.

2. Create a backup of this file.

3. Open the file and search for the **k_variant_op_and** and **k_variant_op_or** values and modify them.

   Example:

```
<key id="k_variant_op_and">AND</key>
<key id="k_variant_op_or">OR</key>
```

Change as follows:

```
<key id="k_variant_op_and">&amp;</key>
<key id="k_variant_op_or">|</key>
```

4. Start Teamcenter and confirm that formula representation is now changed from **AND, OR** to **&, |** successfully.

Enable the tracking of variant changes

To enable the tracking of variant changes, set the **CM_track_variant_condition_changes** preference to **ChangeNotice** types.

When users save changes to a structure containing variants, if this preference is **ON** for the current change type, Teamcenter tracks variant changes between the solution item and the impacted item.

22. Configure modular variants

Create a list of global option definitions

Global options are a convenient way of defining a fixed set of allowed values. Examples of global options are **Color = white, stainless** and **Appliance Width = 500, 600**. You can also set a range such as **Angle >=0, < 360**. You can then reference these global options in other modules.

You can create a list of item identifiers that contain global option definitions and update the **PSM_global_option_item_ids** system preference to include the new item identifier. By default, the list is empty. These definitions can be reused when authoring variant modules.

Example:

PSM_global_option_item_ids=
000400
000410
000420

To include the new item identifier:

1. Create an item to store the global options.
2. Create the desired global options and allowed values.
3. Update the **PSM_global_option_item_ids** preference to include the new item identifier.
4. In Structure Manager, display the **Create Option** dialog box and verify the new item and options are shown in the **Based On** drop-down list.

Enable the creation of variant items of the same type as the parent

You can allow end users to automatically create variant items as the same type as the parent by using the **SaveAs** creation method or link variant items to the generic component by default by setting the **PSECreateVISameType** preference to **true**. By default, it is set to **false**.

Configure modular variants through preferences

Configure modular variants by setting the following preferences.

- **PSEVariantsMode**

  Determines if the modular variants functionality is used. Allowed values are:

- **modular**

  Use only the modular variants functionality. Users cannot view, edit, or configure legacy variants.

- **hybrid**

  Allows users to define modules in structures containing legacy variant data.

- **legacy**

  Use only legacy variants functionality.

The default setting is **PSEVariantsMode=hybrid**.

In addition to enabling the modular variants functionality, the default setting allows users to display, edit, and configure legacy variants, but with the modular variants dialog boxes. Legacy variants do not provide all of the advantages of encapsulation that modular variants enforce. Consequently, Siemens Digital Industries Software recommends you use this setting during migration from legacy variants to modular variants. Once migration is complete, you change the preference setting to **modular**.

- **PSM_global_option_item_ids**

  Set this preference to a list of all item identifiers that contain global option definitions. The default is an empty list. These definitions can be reused when authoring variant modules. For example:

  PSM_global_option_item_ids=
  000400
  000410
  000420

  To implement this functionality:

  1. Create an item to store the global options.
  2. Create the desired global options and allowed values.
  3. Update the **PSM_global_option_item_ids** preference to include the new item identifier.
  4. In Structure Manager, display the **Create Option** dialog box and verify the new item and options are shown in the **Based On** drop-down list.

- **PSESavedConfigRelationTypes**

  When saving a selected option set or variant configuration, the user can choose whether to save to the **Home** folder, the **Newstuff** folder or to the module (item revision) being configured. Use this preference to set the possible relation types from which the user can select when saving to a module.

The default setting is:

PSESavedConfigRelationTypes=
TC_specification
TC_manifestation
TC_relation

- **ShowModuleIcons**

  By default, the icons of the modular variants feature are hidden because their availability affects performance. If you use modular variants, make the icons visible by setting this preference to **true**.

- **PSEAllowLegacyVICreation**

  Set this preference to **true** to permit the creation of variant items for structures that include classic variant options. It also permits the creation of variant items from assemblies that are not modules. The default setting is **false**.

- **PSEBypassVISearch**

  Set this preference to **true** to add a **Create** button to the **Configure** dialog box as soon as the user sets a value for each option. This allows the user to bypass the requirement to search for similar variant item configurations before creating a new variant item. The default setting is **false**.

- **PSECreateVISameType**

  If you set this preference to **true**, Teamcenter always creates a variant item with the same type as its parent and creates the variant item with a **Save As** action. The default setting is **false**.

- **PSEShowUnconfigdVarPref**

  Determines whether structure lines with variant conditions that do not configure for the current variant rule are shown by default in a new Structure Manager window. The preference affects classic variants as well as modular variants. If a structure line is shown even though its variant condition evaluates to **false** for the current variant rule, you can use the following properties to identify the lines that do not configure for the current variant rule:

  ```
  bl_is_variant=Y
  bl_variant_state=
  ```

- **PSEIsNewVILinkedToModule**

  Determines if variant items are linked to the generic item by default. If it is set to **false**, they are not linked to the generic item. The default value of this preference is **true**.

Enable the tracking of variant changes

To enable the tracking of variant changes, set the **CM_track_variant_condition_changes** preference to **ChangeNotice** types.

When users save changes to a structure containing variants, if this preference is **ON** for the current change type, Teamcenter tracks variant changes between the solution item and the impacted item.

Modify AND and OR operators in variant conditions

1. In your Teamcenter installation directory, search for the **tc_text_locale.xml** file, for example, *TC_ROOT***lang\textserver\en_US\tc_text_locale.xml**.

2. Create a backup of this file.

3. Open the file and search for the **k_variant_op_and** and **k_variant_op_or** values and modify them.

   Example:

   ```
   <key id="k_variant_op_and">AND</key>
   <key id="k_variant_op_or">OR</key>
   ```

   Change as follows:

   ```
   <key id="k_variant_op_and">&amp;</key>
   <key id="k_variant_op_or">|</key>
   ```

4. Start Teamcenter and confirm that formula representation is now changed from **AND, OR** to **&, |** successfully.

Set options to copy additional data during variant item creation

By default, arrangements and absolute occurrence data are not copied when a variant item is created to save on system resources.

If you want the system to copy the information when a variant item is created, set the following site preferences to **true**:

Site preference	Description
PSECopyAbsOccDuringVICreation	Controls copy of absolute occurrence data from the generic assembly to the variant item on creation of a variant item.

Site preference	Description
	Valid values: **true** or **false**.
PSECopyArrangementsDuringVICreation	Controls copy of arrangements data from the generic assembly to the variant item on creation of a variant item. This preference is honored only if the preference **PSECopyAbsOccDuringVICreation** is set to **true**. Valid values: **true** or **false**.

For more information about these preferences, see *Administration Data Documentation* on Support Center.

23. Configure incremental changes for the rich client

Enable incremental changes

An incremental change collects together a number of structure changes to a component such as addition or removal of components or changes to attachments (data). Effectivity can be applied to incremental changes to configure the associated changes. This method of change control allows several independent structure changes to be made concurrently, including the addition or removal of unrelated components. Those changes can be implemented in any sequence. This method of change control is suitable for large, flat structures without nested subassemblies or components.

Before enabling incremental changes, you must

- Create the necessary change types by using BMIDE

  When creating incremental change, Teamcenter does not distinguish between change types for change management and incremental change. It is therefore advisable to name the change types clearly.

  For more information about creating the change types by using BMIDE, see *BMIDE for Data Model Design*.

- Provide users with access

  Your can use an **In IC Context rule** to allow Structure Manager or another structure editor application to control access to operations tracked by incremental change. This rule is unlike other rules because it does not depend on the properties of the object.

- Define releases statuses

  Your can set the **Incremental_Change_ReleaseStatus** preference to define the release status.

  Defining a release status in this way allows you to specify effectivity as soon as the incremental change is created. If you do not set this preference, you release the incremental change after it is created. The choice of which of these approaches to implement depends on your business practices. For example, you might set this preference to **Pending IC**.

  The status defined in the preference must already exist. You must also create an access rule that allows write access to objects with this status or you cannot create changes in the incremental change.

> Note:
>
> The status is applied only to incremental change if the effectivity is supplied when it is created.

- Create revision rules

  When you apply a revision rule in Structure Manager, it configures both the structure item revisions and the incremental change revisions. This consideration is important if you use incremental change in combination with traditional revision configuration of structure item revisions. If you want to configure the structure item revisions and incremental change revisions independently, you must apply a separate status.

  The revision rule you create depends on whether the incremental changes are configured by unit or date effectivity, or simply by release date. It also depends on the release status to apply.

  For example, you can apply the **Pre-Released status to the item revisions and the IC in Process status to the incremental change revisions**.

- Allow users to copy or cut structure lines with incremental changes

  By default, if you cut or copy a line and then paste it to a new location, incremental change elements (ICEs) are not copied. This may necessitate significant manual recreation of data if you are cutting or copying many lines together. To automatically copy ICEs, the administrator must set two Business Modeler IDE constants:

  - **Fnd0EnableIceCarryOver**

    Defines item types of the source and destination *parent* BOM line under which ICEs should be copied while copying, cutting, and pasting. For example, by default, the constant value for **Process** is set to **true**, and if the source and destination parent BOM line are both **Process**, Teamcenter copies any ICEs.

  - **Fnd0AttrIcesToExclude**

    Defines the occurrence attributes which Teamcenter does not copy to the target location for occurrence attribute changes.

  For more information about these BMIDE constants, see *BMIDE for Data Model Design*.

  These settings apply to in-context changes to structure lines, their attachments, and their occurrence attributes.

> Note:
>
> For structure changes, only remove changes associated with the original line are carried forward to the new location. Teamcenter does not consider add structure changes; otherwise, two occurrences appear at the same time at two different locations. Teamcenter copies attribute changes made in the immediate parent context, but higher level contexts are excluded. The new ICEs created are associated to the original IC revision.

- Allow users to baseline structures with active incremental changes

  By default, this enhancement is not configured and you must set the following preferences to use it:

  - **IC_baseline_carry_forward_active_changes**

    Set to **True** to carry over active incremental changes when creating a baseline.

  - **IC_baseline_carry_forward_status**

    Defines a list of status names to consider when carrying forward active incremental changes. The default value is **pending**.

Set preferences to enable incremental changes

You can enable incremental changes by setting the following preferences:

Preference	Value
Incremental_Change_Management	Set to **true** to enable incremental changes. Enables incremental change functionality and displays the relevant menu commands. By default, this functionality is disabled. Enabling incremental change using the site preference activates this functionality in all applications where it is available.
EnableIntents	Set to **off** by default as configuration with intents adds a performance overhead if your site does not use intents. If you set this preference to **on**, the intent tabs appear in the appropriate dialog boxes.
ShowUnconfiguredByChangeEffectivity	By default, the unconfigured changes are visible to the user. To view all incremental changes (configured and unconfigured), set this preference to **false**.
Incremental_Change_ReleaseStatus	Defines a release status that is attached to an incremental change when the user first creates

Preference	Value
	it, for example, **Pending**. The status must already exist and have an access rule that allows write access to objects with this status.
MoveICCreationToMenu	Determines if the **Create IC Object** button  on the incremental change toolbar or the **Incremental Change**>**Create Context** menu command is visible. If this preference is **True**, the menu command is visible unless suppressed with Command Suppression; if it is **False**, the button is visible.
IC_baseline_carry_forward_active_changes	Set to **True** to carry over active incremental changes when creating a baseline.
IC_baseline_carry_forward_status	Defines a list of status names to consider when carrying forward active incremental changes. The default value is **pending**.

Control access to incremental changes

To create an incremental change, you must have write access to the incremental change revision. If your site requires a different access policy on incremental changes to regular engineering changes, other rules may be required, for example, the naming convention for incremental change identifiers.

Your administrator can use an **In IC Context** rule to allow Structure Manager or another structure editor application to control access to operations tracked by incremental change. This rule is unlike other rules because it does not depend on the properties of the object. If there is an active incremental change in the structure editor and the operation performed by the user is tracked by incremental change, the **In IC Context (true)** rule is satisfied, and Teamcenter applies its associated ACL. In this situation, applicable structure edits include edits to occurrences, occurrence notes, transforms, and attachments in structure context.

> Caution:
>
> Always use this rule with the **true** argument. The **false** argument applies to all objects, regardless of whether the structure is edited.

Examples of rules to control access to incremental changes

You can place the **In IC Context** rule at the top level of the rule tree as follows:

```
Has Bypass
In IC Context (true) → IC ACL
```

```
Has Status
Has Object
```

In this example, if there is an active incremental change and the operation performed by the user is tracked by incremental change, the **In IC Context** rule applies and its associated named ACL applies to objects affected by the operation. However, if there is no active incremental change, the subsequent rules apply.

You typically use this rule to relax a restriction, such as only allowing edits to a released structure for a specific status. For example:

```
Has Status () → Vault
  Has Status (Pre-Released) → ()   (No ACL)
      In IC Context (true) → Incremental Change Access
```

In addition, the user requires write access to the incremental change itself.

> Note:
>
> There is no mechanism of enforcing that an incremental change is used when making structure edits.

You can also configure the **In IC Context** access rule as follows to allow write access to released structures at a specified status. This allows incremental change to track changes at prereleased stages of the structure (BOM view revision).

```
Has Status () → Vault
  Has Status (Pending IC) → Incremental Change Access (or Working)
  Has Status (Pre-Released) → () (No ACL)
     In IC Context (true) → Incremental Change Access (or Working)
```

The purpose of these rules follows:

- **Has Status ()→Vault**

  This standard rule ensures released parts cannot be modified.

- **Has Status (Pending IC)→Incremental Change Access (or Working)**

  This rule allows you to apply a status of **Pending IC** to an incremental change so that you can apply effectivity. This allows changes to be configured for the incremental change. However, designers still require write access to the incremental change to add changes.

  It may be sufficient to use the general working access control list (ACL). Alternatively, you may want to restrict editing of incremental changes to certain roles, in which case you should use a different ACL (for example, **Incremental Change Access**). When an incremental change is finally released (for example with **Released** status), it can no longer be modified due to the **Has Status ()→Vault** rule.

- **Has Status (Pre-Released)**

  In IC Context (true)→ Incremental Change Access (or Working)

  These rules ensure that when a status of **Pre-Released** is applied to an item revision, changes can only be made to the structure or the item revision with an incremental change. Prior to this, there is no status and the structure or item revision (attachments) can be edited by any user with suitable access permissions.

  When an item revision and the BOM view revision are finally released, for example, with a **Released** status, the structure can no longer be modified due to the **Has Status ()→Vault** rule.

24. Configure the packing of similar structure elements

The packing action groups multiple identical components in one level of an assembly. It groups the components that satisfy the packing criteria, which in turn is configured by setting the following preferences.

- Configure the packing action to exclude or include the sequence number as the packing criteria by specifying the **BOMExcludeFromPackCheck** preference.

  Syntax: **BOMExcludeFromPackCheck**:`seqno|none`

  - If this preference is set to `seqno`, find numbers are excluded from the BOM line pack check. It allows the BOM lines with distinct find numbers to be packed.
  - If the preference is set to `none`, no attributes are ignored during the default pack check. BOM lines must have the same part number and the find number to be packed.

- Specify additional packing criteria by setting the **BOM_Additional_Packing_Criteria** preference.

  Syntax: **BOM_Additional_Packing_Criteria**:`property name`

  - If only the property name is specified, then lines having the same value for the property are packed.
  - If the property name is followed by a colon, the string after the colon is the value when the pack is not allowed.
  - No string after a colon indicates an empty value.

- Configure the structures to load with packed elements by default.

  Syntax: **PSEAutoPackPref**:`0/1`

  - Set the **PSEAutoPackPref** to `1` to load the structures in the *packed* view.
  - Set it to `0` to retain the default view as *unpacked*.

- Configure the reference designator packing rules.

  Syntax: **BOM_Enable_Ref_Designator_Value_Packing**:`True/false`

  Set the **BOM_Enable_Ref_Designator_Value_Packing** preference to configure the reference designator packing rules. You can pack or unpack product structure lines that include reference designators. For example, if **BOM_Enable_Ref_Designator_Value_Packing** is set to `True`, eight BOM

lines with the reference designators **C1, C5, C6, C7, C10, C14, C15, C16** will be packed to one BOM line with the reference designator property **C1, C5-7, C10, C14-16**.

The reference designators are shown in the International Organization for Standardization (ISO) and in the American Society of Mechanical Engineers (ASME) formats. In the ASME format, the reference designator is shown in the following format: uppercase letter followed by a numeral and a letter as a suffix, for example, **S1A**.

25. Configure structure markups

Enable structure markups and set the markup styles

To allow users to work with markups, you must set the **Fnd0BOMMarkupAllowed** global constant to **true** in the Business Modeler IDE.

The properties that are tracked in markups are defined in the **Fnd0BOMMarkupSupportedProperties** global constant in Business Modeler IDE. Properties can be added to this global constant, but the default entries cannot be removed.

You can set the following site preferences to specify how markup actions are displayed:

Site preference	Description
BOM_MARKUP_ADD_FONT_STYLE	Specifies the font style in which added line markups are displayed. For example, if this preference is set to **2**, additions are shown in italic.
BOM_MARKUP_ADD_FOREGROUND	Specifies the color in which added line markups are displayed. For example, if this preference is set to **0,255,0**, additions are shown in green.
BOM_MARKUP_STRIKE_FOREGROUND	Specifies the color in which removed line and property markups are displayed. For example, if this preference is set to **255,0,0**, removals and changes are shown in red.

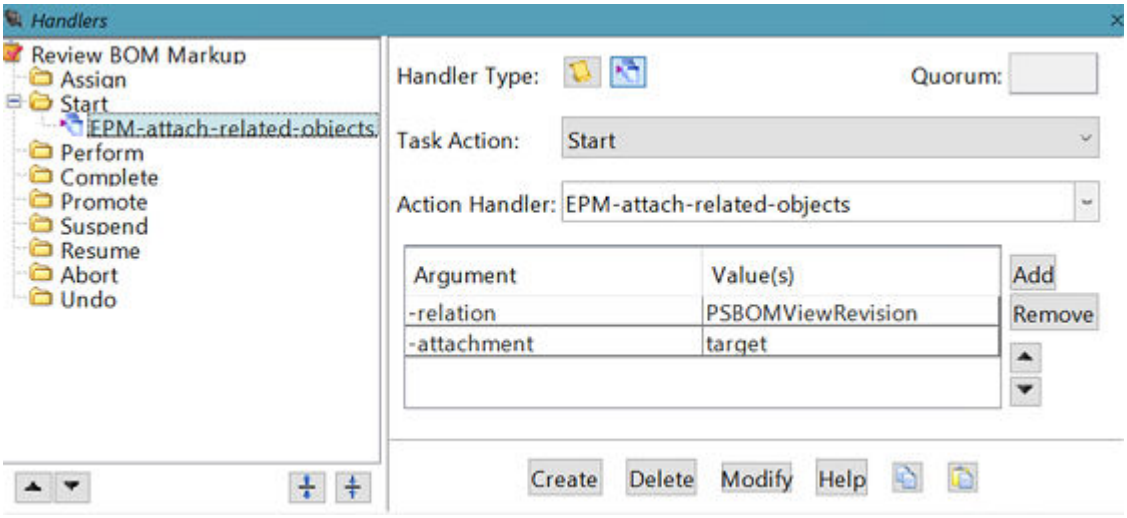
Create a workflow to allow markups for a single object

Use this workflow example if one markup object is related to a BVR and it is the active markup object.

1. Use Workflow Designer to create the following workflow.

2. Open the **Handlers** dialog box and select the **Start** task.

3. Add the **EPM-attach-related-objects** handler, as follows.

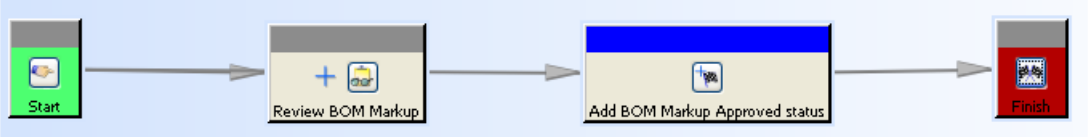


This action adds the markup object as a target. You can send an item revision or BVR to this workflow.

Create a workflow to allow markups for multiple objects

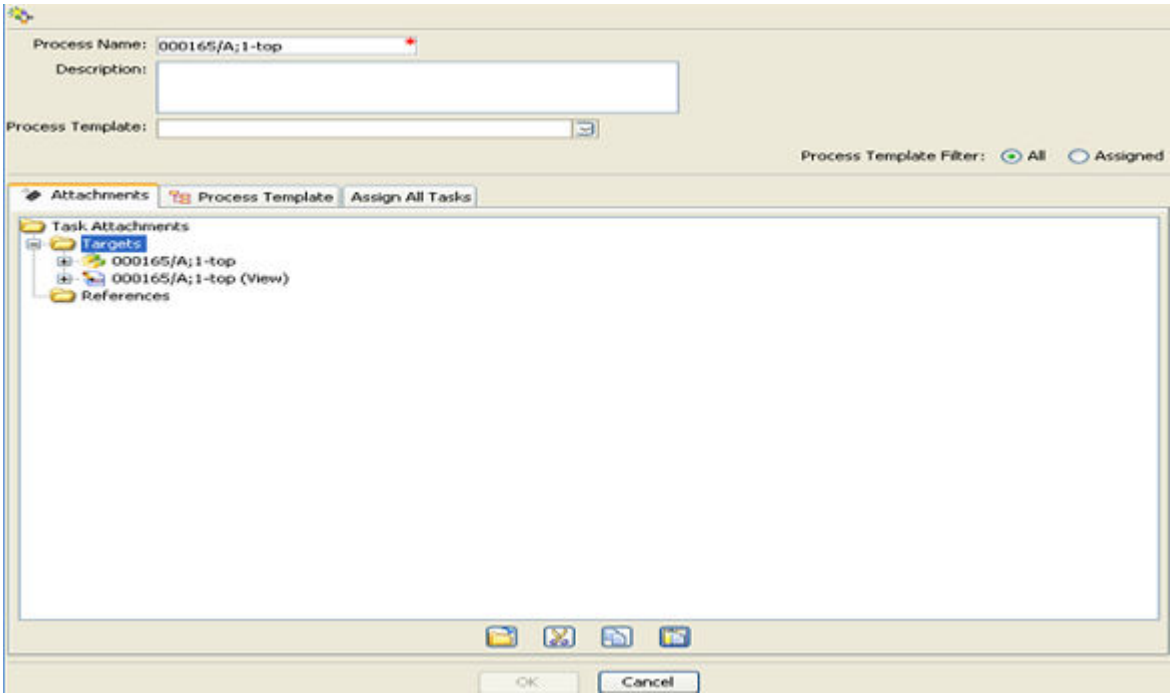
Use this workflow example if more than one markup object is related to a BVR and only one is the active markup object.

1. Use Workflow Designer to create the following workflow.



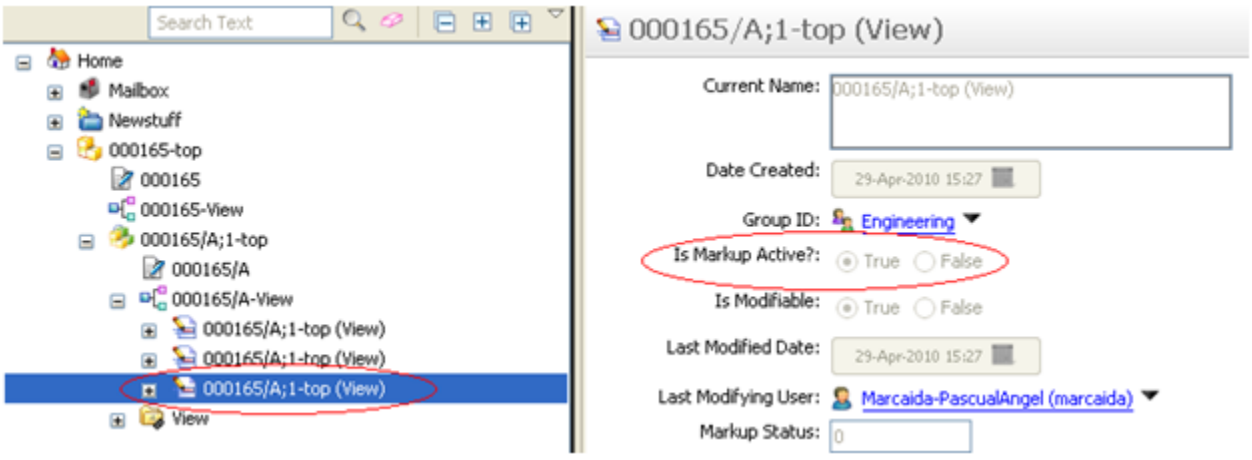
> Note:
>
> Do not add the **EPM-attach-related-objects** handler to the **Start** task because the markup object is a target of the workflow.

2. Copy the active markup and paste it as a specific target of the workflow process.



Create a workflow to carry forward markups for multiple objects

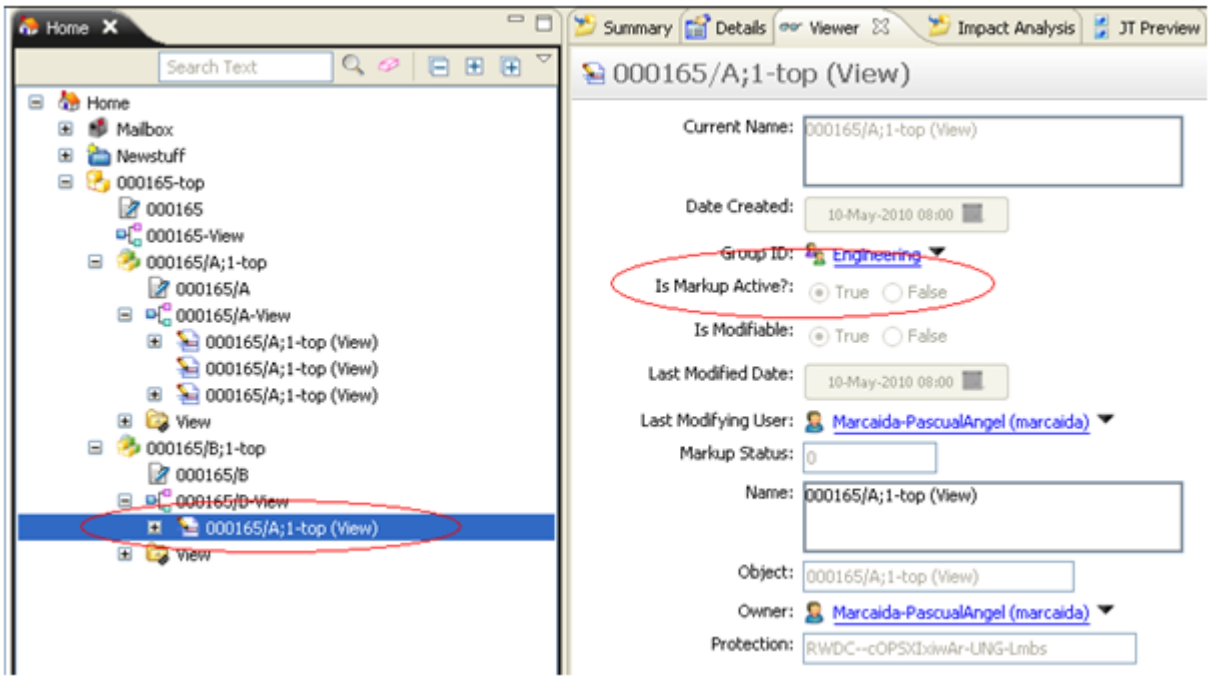
Use this workflow example if more than one markup object is related to a BVR and only one is the active markup object. If you revise the item revision, Teamcenter only carries forward the active markup and you can submit the new revision to the workflow process. For example, the following BVR is related to multiple markup objects, but only one markup object is active.



1. Use Workflow Designer to create the same workflow as described for **sample 1**.

2. Revise the item revision.

   Teamcenter carries forward the active markup and you can submit it to the workflow.



Create a workflow to route markups for review for multiple objects

Use this workflow example when there are multiple markup objects related to a BVR and one object is active. The workflow process filters the active markup and routes it for review. The workflow also applies an approved status to the active markup object.

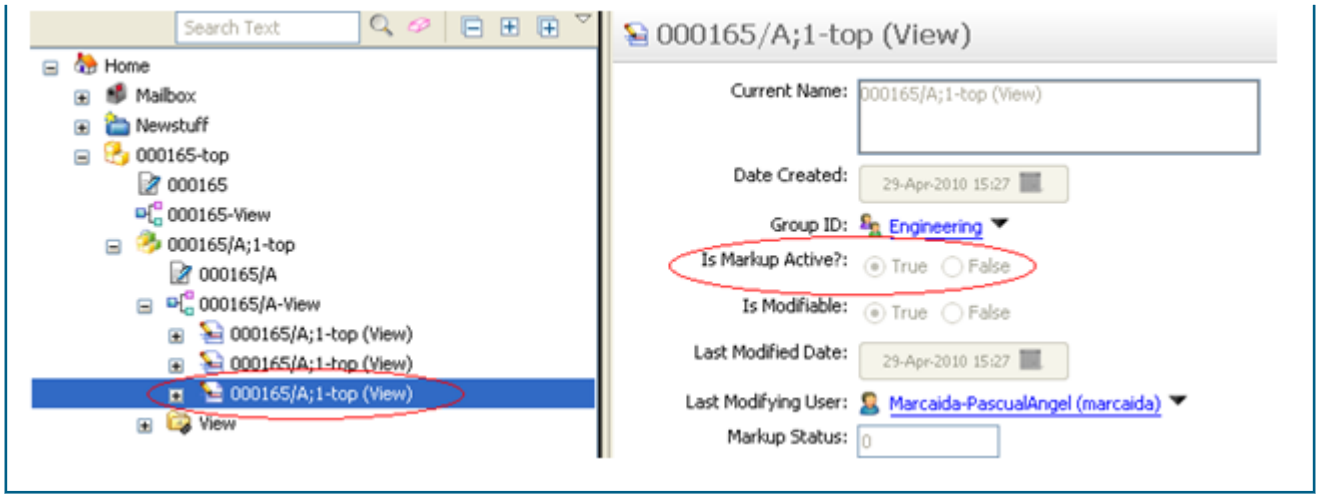
This workflow example includes three processes and they must be invoked in the order listed:

1. **BOMMarkup_Review**
2. **BOMMarkup_SplitTargets**
3. **BOMMarkup_AddStatus**

Use this workflow example if more than one markup object is related to a BVR and only one is the active markup object. It provides a workflow example that accepts an item revision or BVR with multiple markup objects for review, but only adds a **BOM Markup Approved** status to the active markup object.

If you select the item revision as the initial target, Teamcenter adds **BOM Markup Approved** status to the item revision, BVR, and the active markup objects. If you select the BVR as the initial target, Teamcenter adds **BOM Markup Approved** status to the BVR, and the active markup objects.

> Note:
>
> A BVR may contain several markup objects, but only one of these objects can be active, as shown in the following example.

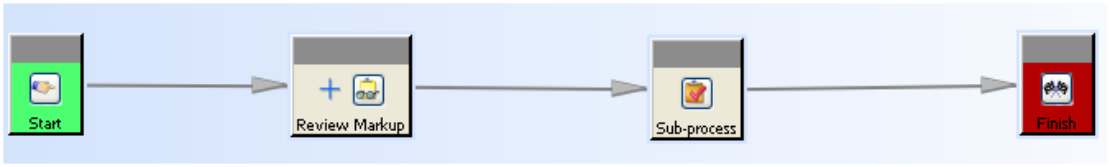


BOMMarkup_Review process

Sends an item revision or BVR to a review task. After approval, it starts the **BOMMarkup_SplitTarget** process.

The review task is a manual operation. The reviewer or approver must send the workflow target (the item revision or BVR) to Structure Manager to view the product structure. Markup mode must be switched on to see the proposed changes.

1. Use Workflow Designer to create a new **BOMMarkup_Review** process. Place a review task and a generic task between the **Start** and **Finish** tasks, as follows.



2. Select the **Start** task and click the **Task Handlers** button .

   Teamcenter displays the **Handlers** dialog box.

3. Verify that the **EPM_attach-item-revision-targets** handler is attached to the **Start** task action.

4. Select the **Sub-process** task and click the **Task Handlers** button .

   Teamcenter displays the **Handlers** dialog box.

5. Add the **EPM-create-sub-process** under the **Complete** task action and specify the following arguments and values:

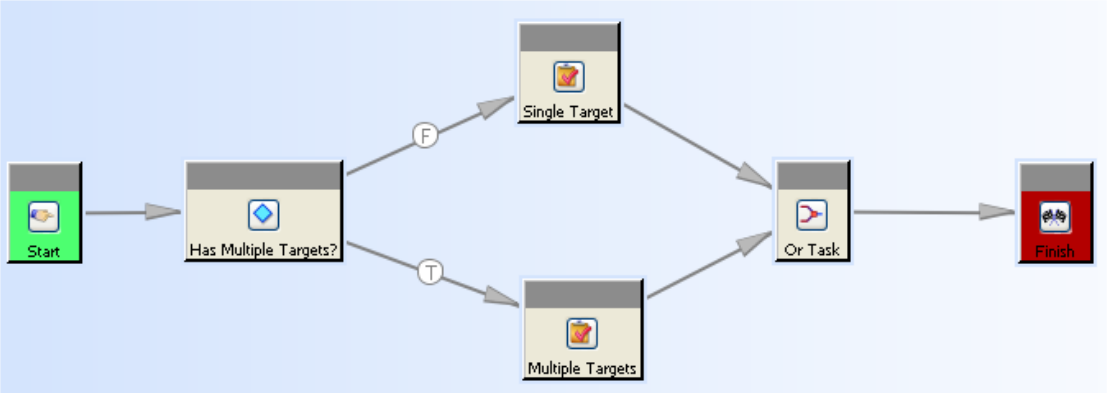
Argument	Values
-template	BOMMarkup_SplitTargets
-from_attach	TARGET
-to_attach	TARGET

6. Make the template available for use by selecting the **Set stage to available** check box.

BOMMarkup_SplitTargets process

Adds the markup objects as targets. As there may be multiple markup objects, the workflow sends each of them individually to the **BOMMarkup_AddStatus** object to identify the active markup.

1. Use Workflow Designer to create a new **BOMMarkup_SplitTargets** process. Add a condition task, two generic tasks, and an **OR** task between the **Start** and **Finish** tasks, as follows.



2. Select the **Start** task and click the **Task Handlers** button.

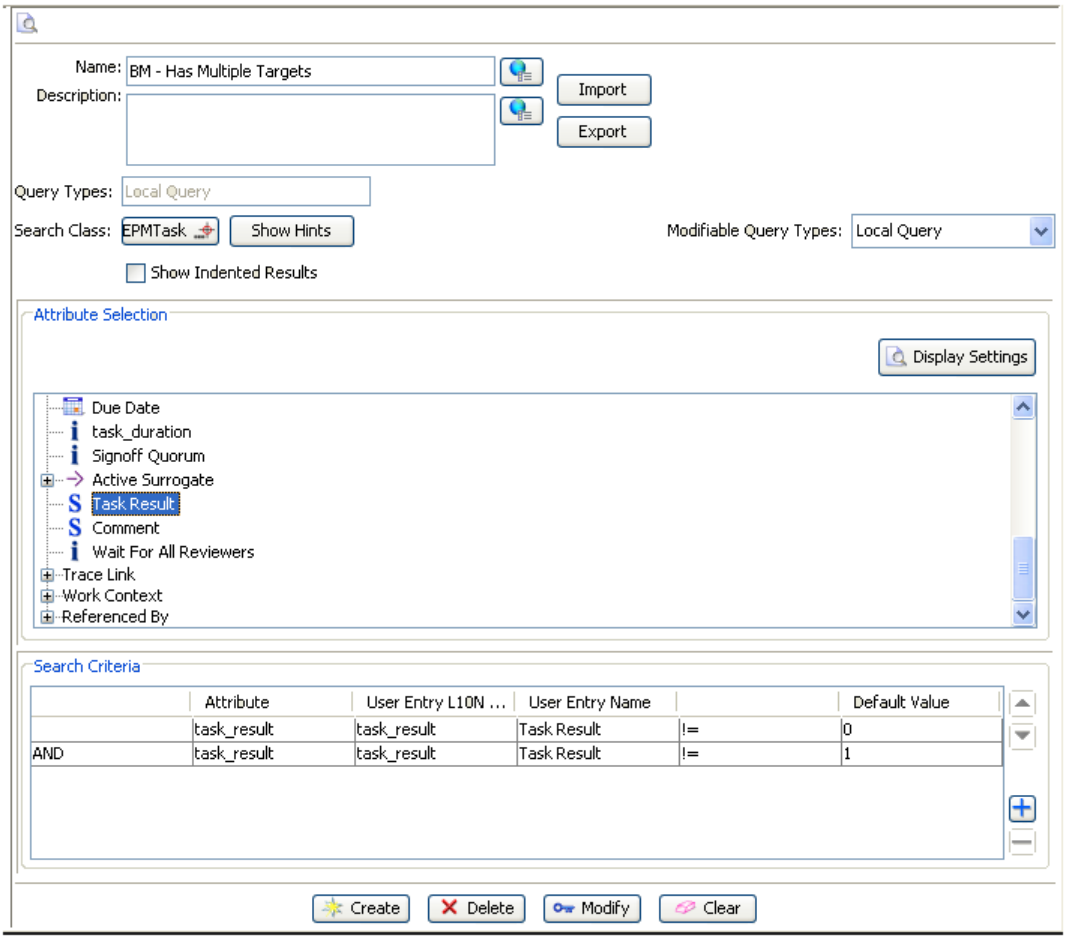
   Teamcenter displays the **Handlers** dialog box.

3. Verify that the **EPM-attach-related-objects** handler is defined under **Start** task action as follows. This action adds the markup objects as workflow targets. The arguments and values are as follows:

Argument	Values
-relation	Fnd0PS_Markup
-att_type	target

4. Create a **Has Multiple Targets?** condition task that counts the number of workflow target objects and determines if there is more than one. Create this condition task with workflow handlers and a query object, as follows:

   a. In Query Builder, create the following query to check the value of the task result.



b. In Workflow Designer, select the **BOMMarkup_SplitTargets** template, and then select the **Has Multiple Targets?** condition task.

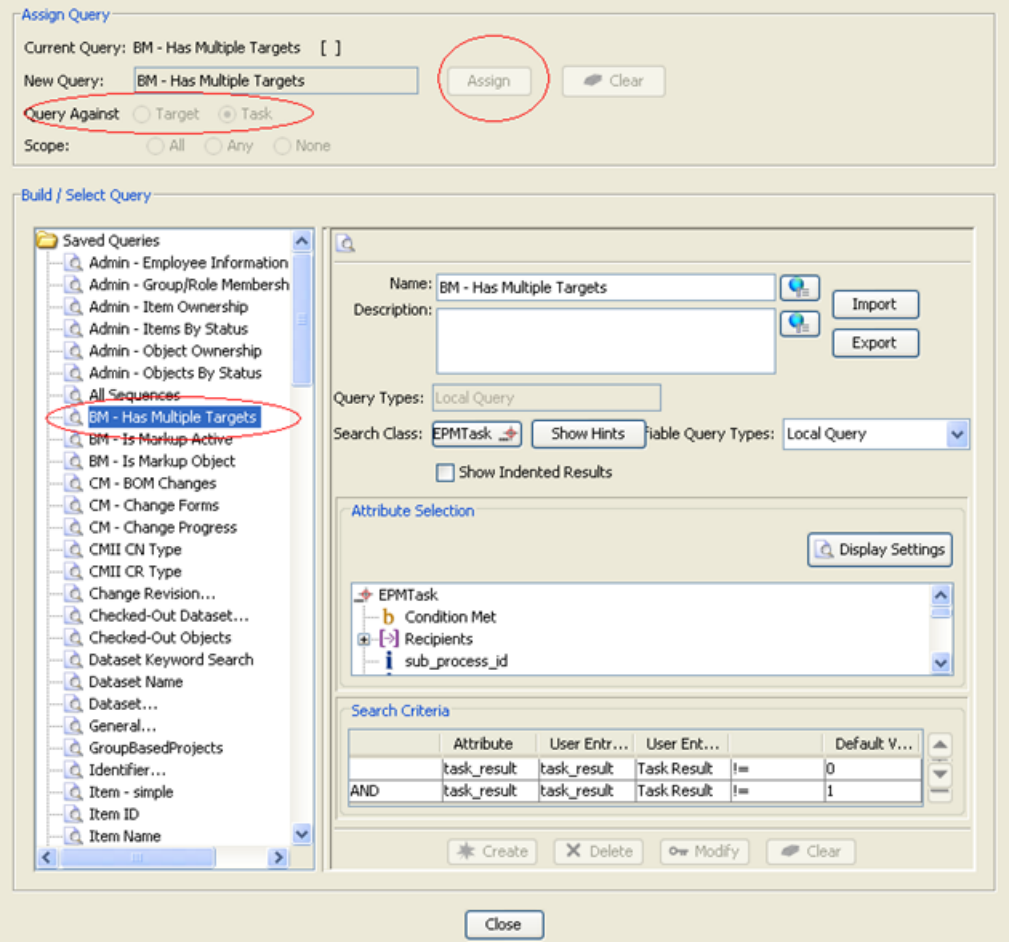
c. Click the **Attributes** button .

Teamcenter displays the **Attributes** pane.

d. Click **Condition Query**.

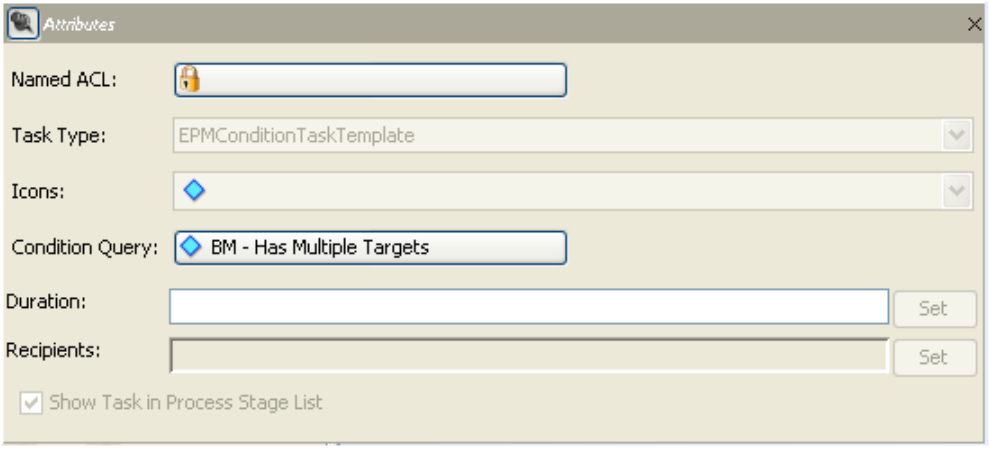
Teamcenter displays the **Condition Query** dialog box.

e. Select **BM-Has Multiple Targets?** and **Query Against - Task** as shown, and then click **Assign** to assign the query.



f. Close the dialog box.

g. Click **Display Settings** and ensure the **Attributes** dialog box shows the following settings.



h. Close the dialog box.

i. Ensure the **Has Multiple Targets?** condition task is still selected and click the **Task Handlers** button.

Teamcenter displays the **Handlers** dialog box.

j. Add the **EPM-set-task-result-to-property** handler under **Start** task as follows. This reads the **num_property** task property and sets its value as the task result. Arguments and values are as follows:

Argument	Value
-property	**num_targets**
-source	**task**

k. Verify that the **EPM-set-condition** handler is defined under **Start** as follows. (This handler was added in step b previously.) Arguments and values are as follows:

Argument	Value
$Query	**BM - Has Multiple Targets**
-query_type	**Task**

5. Configure the **Single Target** task to start the **BOMMarkup_AddStatus** subprocess. Select **Single Target** and click the **Task Handlers** button.

Teamcenter displays the **Handlers** dialog box.

6. Create the **EPM-create-sub-process** handler under the **Complete** task. Arguments and values are as follows:

Argument	Value
-template	**BOMMarkup_AddStatus**
-from_attach	**TARGET**
-to_attach	**TARGET**

7. Configure the **Multiple Targets** task to start the **BOMMarkup_AddStatus** subprocess for each target. Select **Multiple Targets** and click the **Task Handlers** button.

Teamcenter displays the **Handlers** dialog box.

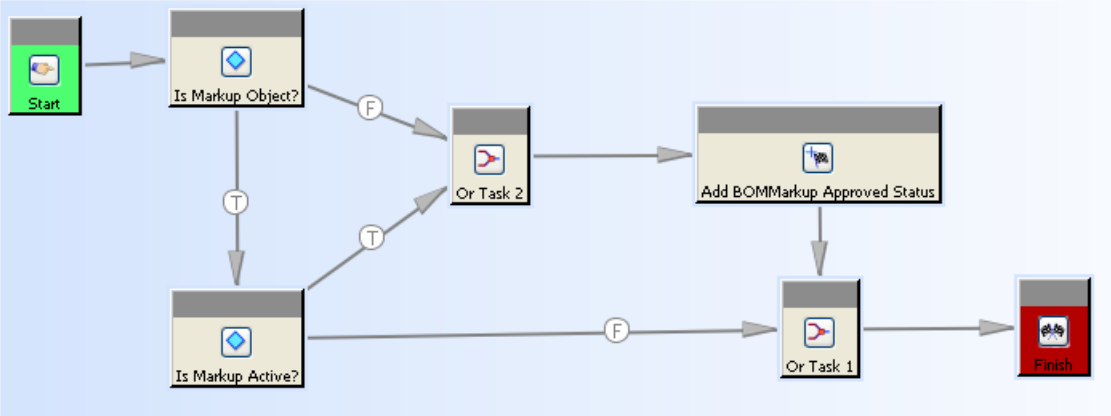
8. Create the EPM-create-sub-process handler under the **Complete** task.

9. Make the template available for use by selecting the **Set stage to available** check box.

BOMMarkup_AddStatus process

- Always has a single target object because the parent **BOMMarkup_SplitTargets** process always spawns a new **BOMMarkup_AddStatus** subprocess for each new target object.
- Checks if the target object is a markup object. If not, it must be an item revision or BVR, and Teamcenter sends it to the **Add BOMMarkup Approved Status** task.
- If the target is a markup object, checks if it is active. If so, Teamcenter sends it to the **Add BOMMarkup Approved Status** task; otherwise, no status is added.

1. Use Workflow Designer to create a new **BOMMarkup_AddStatus** task. Add two condition tasks, two **OR** tasks, and an **Add Status** task between the **Start** and **Finish** tasks, as follows.



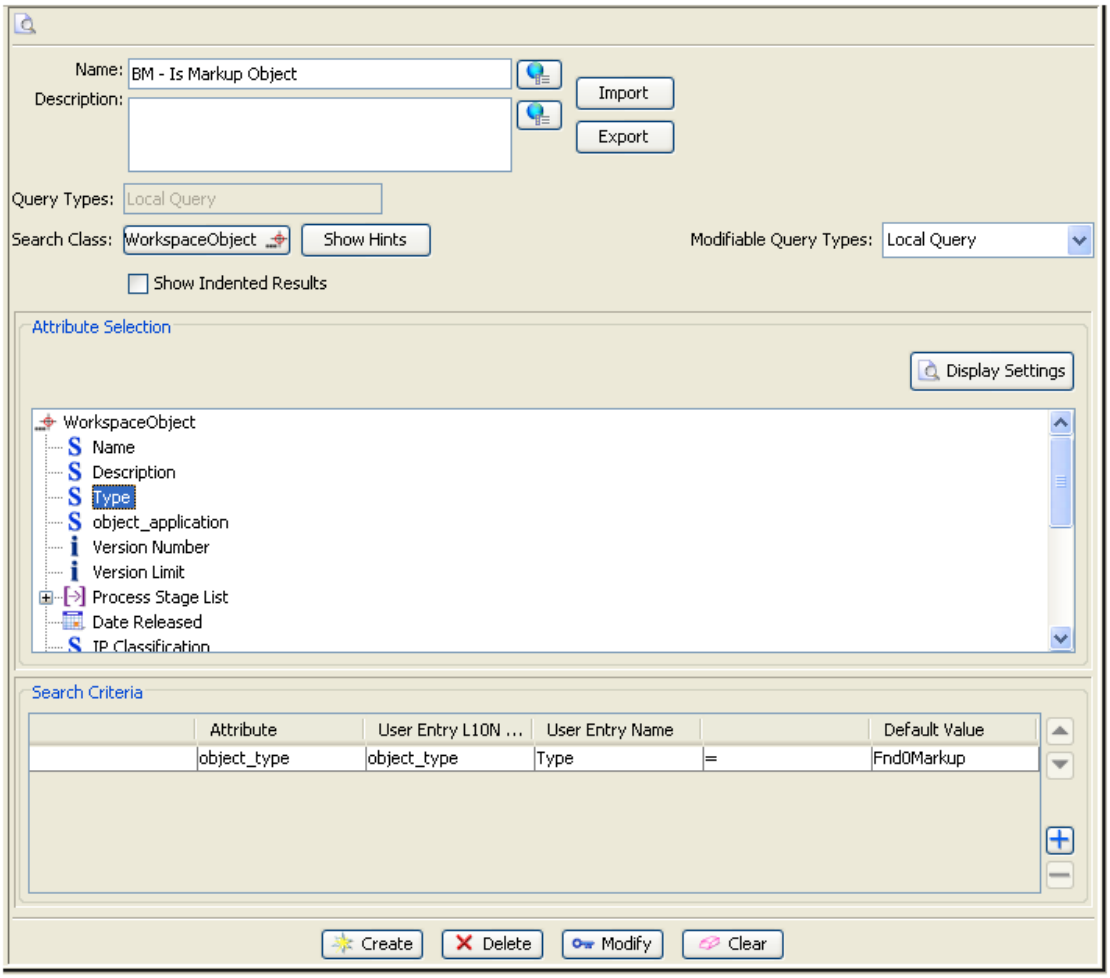
2. Select the **Start** task and click the **Task Handlers** button.

   Teamcenter displays the **Handlers** dialog box.

3. Verify that the **EPM-attach-related-objects** and **EPM-attach-item-revision-targets** handlers are *not* defined under the **Start** task. It is not necessary to add or introduce more targets to the subprocess.

4. Create a **BM – Is Markup Object?** condition task that checks if the target object is a markup object. (The process expects only one target object.) It does this by checking the **object_type** property value of the markup object with a query that you create as follows.

   a. In Query Builder, create the following query.



b. In Workflow Designer, select the **BOMMarkup_AddStatus** template, and then select the **BM – Is Markup Object?** condition task.

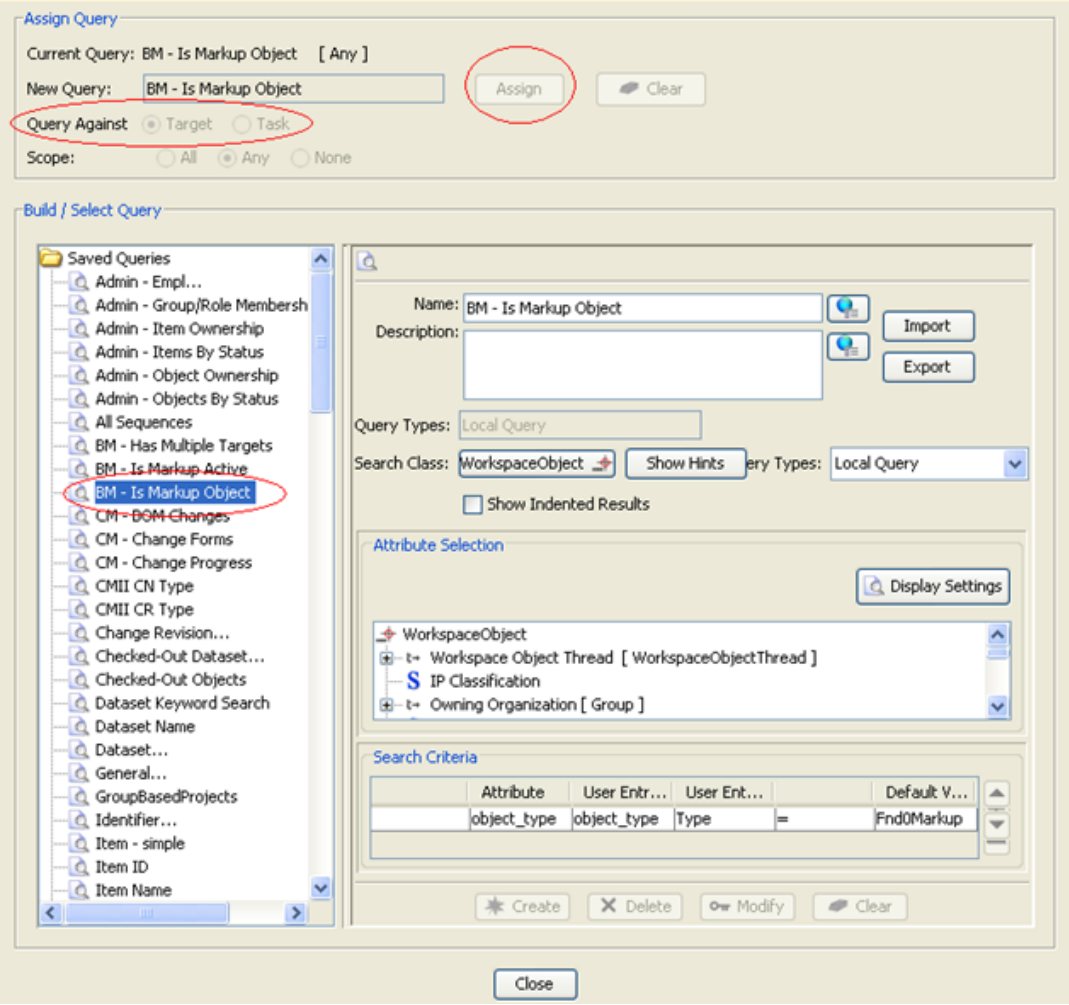
c. Click the **Attributes** button .

Teamcenter displays the **Attributes** pane.

d. Click **Condition Query**.

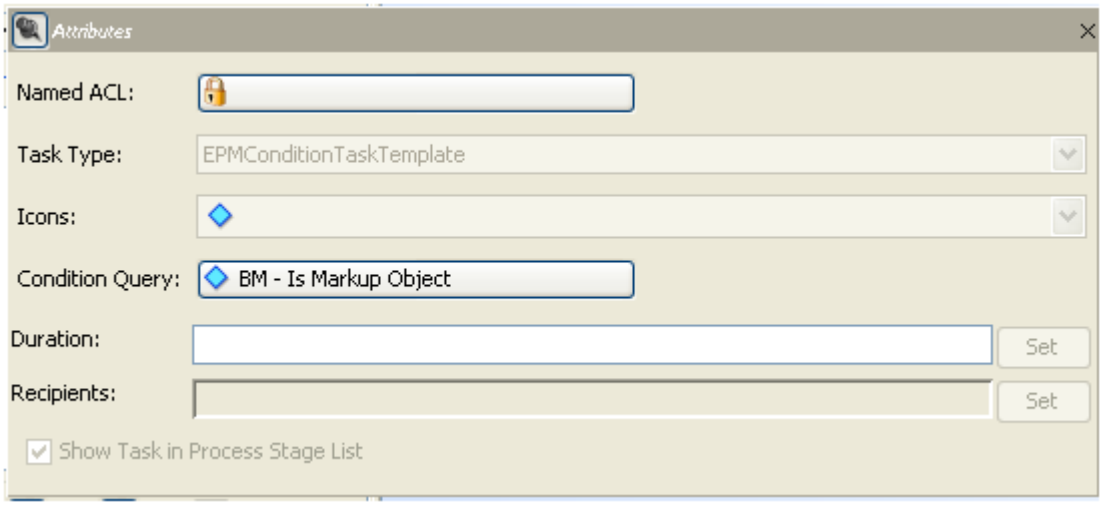
Teamcenter displays the **Condition Query** dialog box.

e. Select **BM-Is Markup Object?** and **Query Against - Task** as shown, and then click **Assign** to assign the query.



f. Close the dialog box.

g. Click **Display Settings** and ensure the **Attributes** dialog box shows the following settings.



h. Close the dialog box.

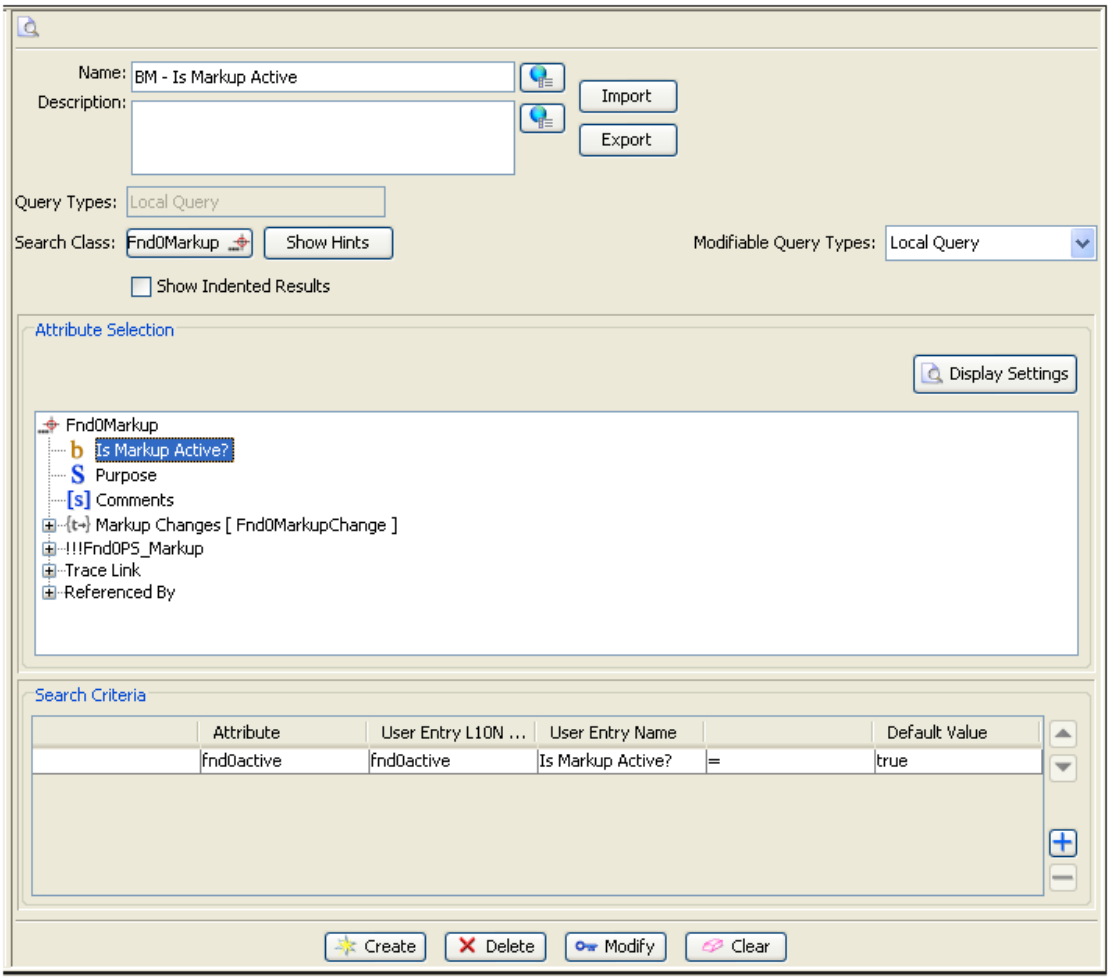
i. Ensure the **BM – Is Markup Object?** condition task is still selected and click the **Task Handlers** button.

Teamcenter displays the **Handlers** dialog box.

j. Verify that the EPM-set-condition handler is defined under the **Start** task.

5. The **BM – Is Markup Active?** condition task checks if the markup object is active. It does this by querying the **fnd0active** property value of the markup object.

a. In Query Builder, create the query as follows.



b. In Workflow Designer, select the **BOMMarkup_AddStatus** template, and then select the **BM – Is Markup Active?** condition task.

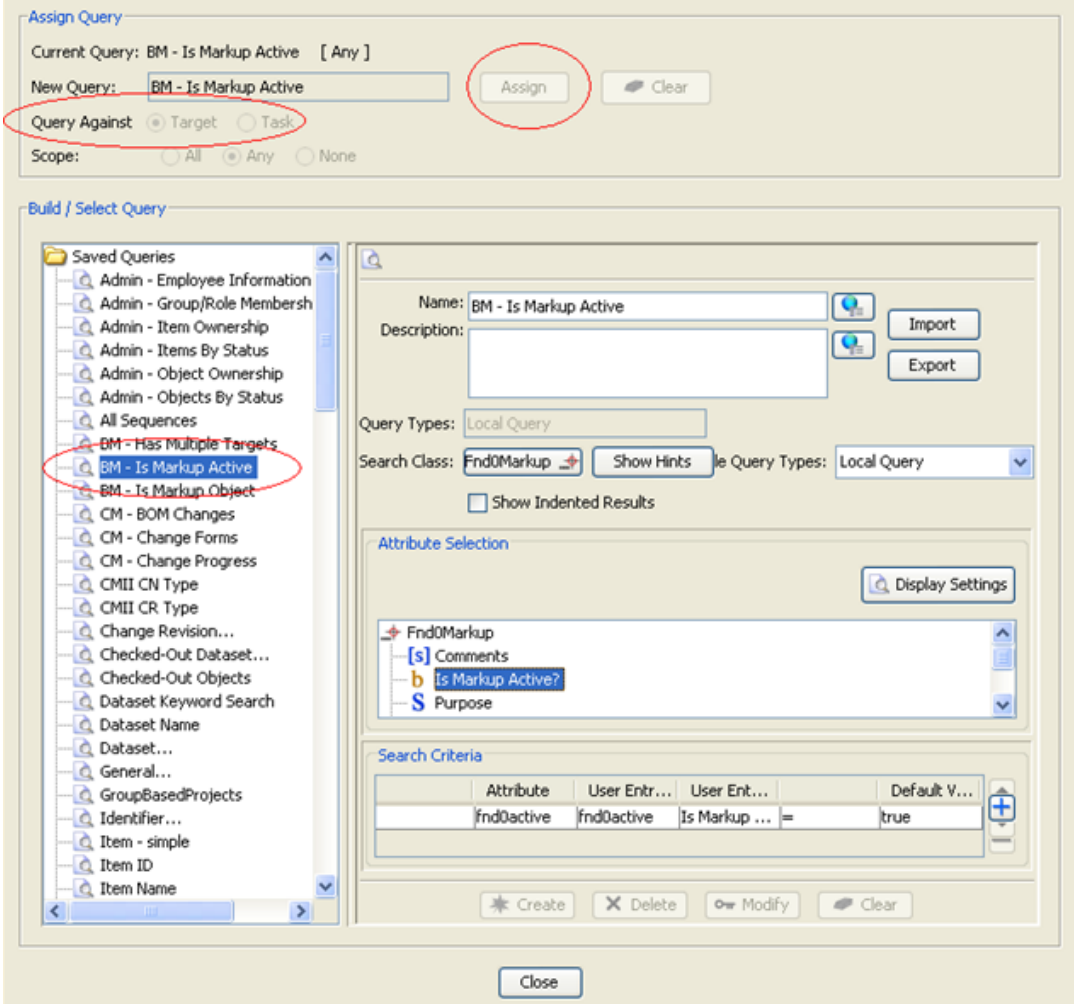
c. Click the **Attributes** button.

Teamcenter displays the **Attributes** pane.

d. Click **Condition Query**.

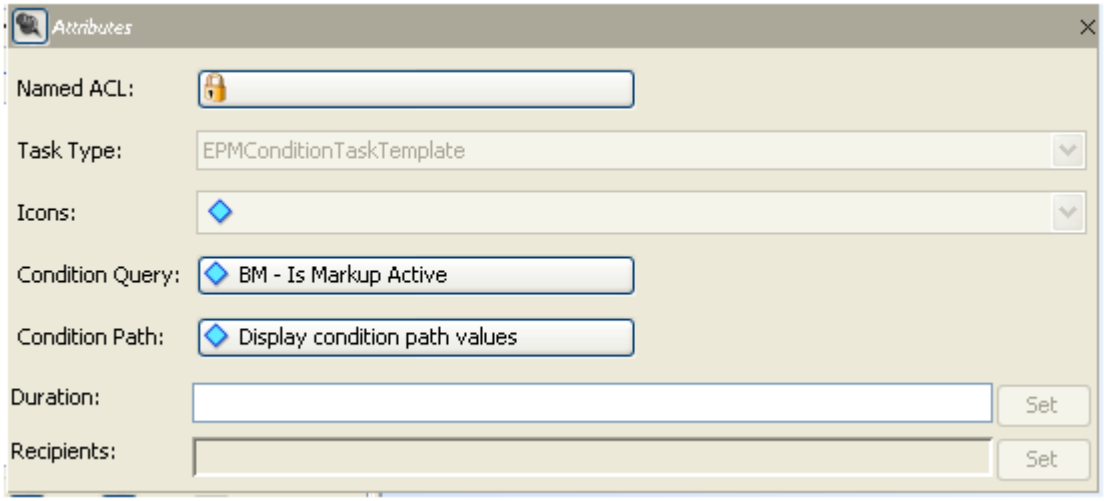
Teamcenter displays the **Condition Query** dialog box.

e. Select **BM – Is Markup Active?** and **Query Against - Target** as shown, and then click **Assign** to assign the query.



f. Close the dialog box.

g. Click **Display Settings** and ensure the **Attributes** dialog box shows the following settings.



h. Close the dialog box.

i. Ensure the **Is Markup Object?** condition task is still selected and click the **Task Handlers** button .

Teamcenter displays the **Handlers** dialog box.

j. Verify that the EPM-set-condition handler is defined under the **Start** task.

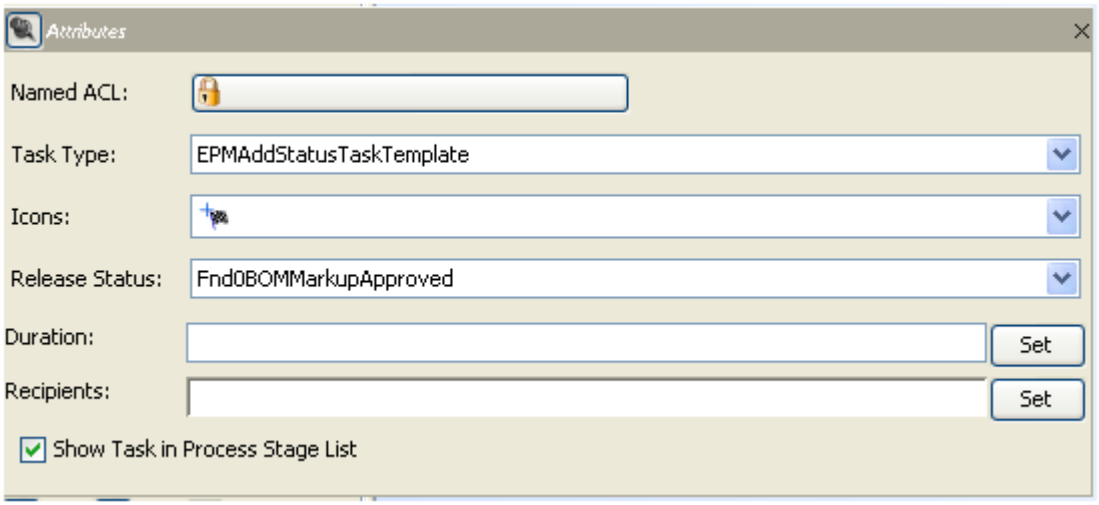
6. The **Add BOMMarkup Approved Status** condition task adds the **BOMMarkup Approved** status to the target object, which may be an item revision, BVR, or active markup object.

> Note:
>
> **BOMMarkup Approved** is not a standard status type. It is created for this workflow to ensure the system can write to the target objects of the workflow after the approval process.

a. In Workflow Designer, select the **Add BOMMarkup Approved Status** status, and then click the **Attributes** button .

Teamcenter displays the **Attributes** pane.

b. Enter a **Release Status** of **Fnd0BOMMarkupApproved** as follows.



c. Ensure the **Add BOMMarkup Approved Status** status is still selected and click the **Task Handlers** button.

Teamcenter displays the **Handlers** dialog box.

d. Verify that the EPM-create-status handler is defined under the **Start** task.

7. (Optional) In Access Manager, create a write operation in the rule tree for **status=BOMMarkup Approved**. This setting allows you to apply markups to the product structure after the review and approval process.

8. Make the template available for use by selecting the **Set stage to available** check box.

Create a workflow to review and apply BOM markups

For users to review and apply BOM markups in the context of change, you must create a workflow process template by using the following workflow handlers:

- **CM_Approve_ECO_Markup_Handler**
- **CM_validate_ECO_Markup_Handler**
- **CM_Cancel_ECO_Markup_Handler**

For more information on creating a workflow process template, see *Workflow Designer on Rich Client*.

26. Configure structure comparison for Active Workspace

About configuring structure comparison

Whenever two structures are compared, equivalent occurrences between the two structures are identified first. The system uses the **ID In Context** property value to identify equivalent occurrences. Properties other than **ID In Context** can also be used to identify equivalent occurrences in the structures.

What happens when Dynamic Equivalence check box is not selected?

The system performs an accountability check to compare the two structures. *Accountability check* is the verification mechanism to check if all elements in one structure have equivalent elements in other structure. For users to perform an accountability check, ensure that the Multi-Structure Foundation license is obtained and installed in your Teamcenter setup.

While performing an accountability check, the system first compares the **Id In Context** property.

If **Matched** option is selected in comparison results, those properties that have same **Id In Context** property will be shown in the comparison results as matching.

If **Different** option is selected, then the properties that have matching **ID In Context** property are further compared based on the properties specified in the **AWBPropertiesToCompare.BVR** preference. The elements that have different values for the compared properties are highlighted as partial matches.

What happens when Dynamic Equivalence check box is selected?

The system compares the structure elements based on the properties specified in the **MEAccountabilityCheckDynamicIDICProperties** preference. By default, this preference contains the **Item ID** property. Any other property added to this preference will be included in the comparison.

> Example:
>
> If the default **Item ID** property is used as the equivalence criteria, the lines with the same **Item ID** are treated as equivalent.

When the **Dynamic Equivalence** check box is selected, the **ID In Context** property is ignored unless it is added to the **MEAccountabilityCheckDynamicIDICProperties** preference.

After the dynamic equivalence check, the equivalent lines are further compared to find if it is a partial match or a full match. Properties for this comparison are specified in the **AWBPropertiesToCompare.BVR** preference. By default, this preference contains **Find No.** and **Item revision ID**.

MEAccountabilityCheckDynamicIDICProperties

The following example shows the **MEAccountabilityCheckDynamicIDICProperties** preference syntax and description.

Example:

```
<preference name="MEAccountabilityCheckDynamicIDICProperties"
type="String" array="true" disabled="false" protectionScope="Site"
envEnabled="false">

<preference_description>
Defines the list of BOM Line properties used by accountability check
dynamic equivalence criteria when searching for equivalent BOM Lines.
By default the BOM Line property "bl_item_item_id" is always included
in dynamic equivalence criteria, unless certain structure search
friendly properties are provided:
- internal BOM Line properties pointing to Occurrence Note
- compound property on a Form
- property on an Item type, description or name
(e.g. "bl_item_object_type")
- property on an Item Revision type, description or name
(e.g. "bl_rev_object_name")
If none of these is available, the property "bl_item_item_id" will
be used in conjunction with any other specified properties. The
first cacheless(structure) search property in the preference is
used as an initial property to collect the BOM Lines on which the
other properties are evaluated. Empty property values are not
considered as equal during compare. As an example for a value to
search for absolute transformations, the entry "bl_plmxml_abs_xform"
can be added.
 </preference_description>
      <context name="Teamcenter">
           <value>bl_item_item_id</value>
      </context>
    </preference>
```

If you specify a different value (other than the default one, **bl_item_item_id**), an accountability check is performed to compare two structures. Accountability check requires the Multi-Structure Foundation license.

Specify comparison properties and comparison result retention period

For comparing two structures and for comparing two elements within a structure, you must specify the **BOMLine** properties to be compared in the **AWBPropertiesToCompare.BVR** preference.

Additionally, you can define for how long the comparison results can be retained in the **Awb0CompareResultCleanupDays** preference. After this time period, the comparison results are deleted.

27. Set up the BOM view

About BOM view types

The *view type* is an attribute of a BOM view revision that indicates its purpose, for example, design or manufacturing. It allows you to distinguish one BOM view revision to be distinguished from another BOM view revision in the same item revision.

The view type of a BOM view revision in a parent assembly is not dynamically configured in the same way as revisions. You must specify the view type when you add the component to the assembly, for example, adding a specific (nonconfigurable) view of the component. If the component has multiple views, you can change the view type referenced by the parent assembly, but must have write access to the parent assembly to complete this edit operation.

The administrator defines a set of view types for each site using the Business Modeler IDE. Any number of view types can be defined, but most sites only require a single view type for product structure synchronization with Structure Manager.

Create a BOM view type

1. Start the Business Modeler IDE and, in the Extensions view, select the project in which you want to create the new note type. Right-click the project and choose **Organize→Set active extension file** on the shortcut menu. Select the **options.xml** file as the file in which to save the new BOM view type.

2. Expand the project and the **Options→View Type** folders.

3. Right-click the **View Type** folder and choose **New View Type** from the shortcut menu. The **New View Type** wizard runs.

4. Perform the following steps in the **View Type** dialog box:

   > Note:
   >
   > The **Project** box defaults to the already selected project.

   a. In the **Name** box, enter the name you want to assign to the new BOM view type.

   b. In the **Description** box, enter a description of the new BOM view type.

   c. Select the **Attach Value List** check box if you want to attach a list of values (LOV) to the BOM view type.

   d. If you select the **Attach Value List** check box, click the **Browse** button to the right of the **LOV** box to locate the list of values to attach to the BOM view. Type an asterisk * in the **Find** dialog

box to see all possible selections. Click the **Browse** button to the right of the **Default Value** box to choose the value from the list of values that you want to use for the BOM view type.

e. Click **Finish**.

The new BOM view type displays under the View folder in the Extensions view.

5. To save the changes to the data model, choose **File →Save Data Model**.

Set the default view type

This option sets the default view type Teamcenter uses when multiple views are present and there is ambiguity, for example, when opening Structure Manager or pasting in an assembly.

Typically, your administrator sets a default view type for all users at the site, for example, **Design**. However, this setting is most appropriately defined in conjunction with each user's role. For example, users with the role of production engineer can define their default view type as **Manufacture**; users with the role of designer can define their default view type as **Design**.

- Corresponding preference name: **PSE_default_view_type**
- Default setting: **view**

Display the view type attribute in the BOMLine title

Set the **PSEShowViewTypePref preference** to **True** to display the **View Type** attribute in the **BOMLine** title. Set the preference to **False** to hide the **View Type** attribute.

Set access control on view types

You can use Access Manager to create rules that control write access to the different view types, preventing users from inadvertently editing the structure when manipulating multiple views. For example, production engineers creating manufacturing views should not have write access to design view types. This would prevent production engineers from unintentionally cutting components from the design view when copying components from the design view to create the associated manufacturing view.

BOM views and arrangements

If you use assembly arrangements in Teamcenter (typically by synchronizing arrangement data between NX and Teamcenter) and want to see all assembly arrangements displayed as children of their associated BOM view revisions in My Teamcenter, you should modify the value of the **BOMView Revision_DefaultChildProperties** preference by adding **Fnd0AsSavedArrangement** to the list of values.

If you do not use assembly arrangements in Teamcenter, or use them but do not wish to see them displayed as children of BOM view revisions in My Teamcenter, it is not necessary to modify this preference.

Sites with multiple BOM views

Sites with multiple BOM views

The creation of multiple views is discouraged for new Teamcenter deployments but continues to be supported for existing customers with successful deployments. Siemens Digital Industries Software therefore no longer recommends that you attempt to transition an existing site to multiple views. Several possible alternatives to multiple views are available.

Although Teamcenter supports multiple BOM views, the default configuration is a single BOM view type called **View** that applies to the entire site. To manage multiple BOM views, you must designate one BOM view type as the default view by setting the **PSE_default_view_type** preference. Because the initial (default) configuration of Teamcenter installation and upgrades from all previous versions of Teamcenter is single view and uses the **View** BOM view type, this preference is set to **view** by default.

Rename the default view

When sites transition to multiple BOM views, the **View** BOM view name may be too generic to retain. Before proceeding with a site-wide implementation of multiple BOM views, consider if you want to continue using this generic BOM view type or if you should rename it to something more meaningful.

When you rename a BOM view type, the change is immediate. However, if you use the default BOM view revision (BVR) names provided with Teamcenter, these names are no longer appropriate. When it creates each BVR, Teamcenter assigns it a name according to the BOM view type and item revision ID. If you subsequently change the name of the BOM view type, the names of existing BOM view revisions that use this name are not automatically updated. You can rename affected BVRs by running the **ps_rename_bvrs** utility against them.

The **ps_rename_bvrs** utility renames existing BOM views and BOM view revisions (BVRs) to the current default naming scheme implemented by the **USER_ps_default_bom_view_name** and **USER_ps_default_bvr_name** user exits. You do not need to synchronize BVR names immediately after renaming the BOM view type, but can perform this activity in batches for multiple item identifiers.

Prerequisites for implementing multiple BOM views

Before implementing multiple BOM views, you should do the following:

- In the default configuration, all data is stored in the **View** BOM view. If you decide to rename this default view, choose a new name that reflects the primary use of this view, for example, **Design**.

- Establish BOM view naming conventions before you create additional product structure data. This minimizes the number of BVRs that you must rename to reestablish automatic updates.

Rename multiple BOM views

To transition a site to multiple BOM views by renaming the default BOM view type:

1. Rename the existing default **View** BOM view type to your new selected default name.

2. Decide if this renamed BOM view type is the new default BOM view type. In most cases, you would make it the default because your existing product structure data is saved under it. However, you can make any valid BOM view type the default view.

3. If you decide to make *another* BOM view type the default view, create the necessary BOM view type.

4. Change the **PSE_default_view_type** preference setting to the new default BOM view type.

5. Decide if the BOM view renamed in step 1 is synchronized to NX. If so, change the **TC_NX_view_type** preference to the renamed BOM view type.

6. Run the **ps_rename_bvrs** utility against the affected BOM view revision names.

28. Enable visualization for structures

Enable product visualization through preferences

Users can visualize a product structure either in the embedded viewer or in the stand-alone Teamcenter lifecycle visualization product. For this, you must enable the product visualization capability by setting the following preferences:

Preference	Value
TC_show_open_in_vmu_button	Set the value to **true** to make the **Open in Lifecycle Visualization** button and menu command visible to users.
TC_Schematic_BOMView_types	Set the value to the required BOM View type so that users can view and mark up schematic diagrams, including electrical routing, hydraulic, and piping diagrams. Example: For viewing schematics of a piping system, set the value to **Piping System View**.
TCVIS_allow_NGC_expand_mode	By default, a Non Geometry Component (NGC) attached to a BOM line is expanded and displayed in the viewer. To disable the expansion, set the value to **0**.
TCVIS_enable_NGC_feature	Set the value to **false** to disable the expansion and display of NGCs. This value of this preference overrides the value of **TCVIS_allow_NGC_expand**.

Manage unconfigured data in a product view

Users can save data in the assembly viewer, including current items, zoom factor, rotation angle, and pan displacements in a *product view* (sometimes called a *3D snapshot or snapshot view*). You can set the following preferences to determine how the system creates a product view when unconfigured objects are shown:

Preference	Description
Vis_PV_InvalidConfigWarnLevel	If set to **Warning**, Teamcenter displays a warning message if any of the options specified in the **Vis_PV_BlockingViewToggles** preference are on, but you can still create or update a product view. If set to **Prevent**, Teamcenter prevents you creating or updating a product view if any of the view configuration options specified in the **Vis_PV_BlockingViewToggles** preference are on. If set to **Off**,

Preference	Description
	the state of the menu commands does not affect whether product views are created or updated. You can also set this preference with the **Product View Creation Preferences→View Toggle Warning Level** option.
Vis_PV_BlockingViewToggles	Specifies the view states that are evaluated when the **Vis_PV_InvalidConfigWarnLevel** preference is set to show a warning or prevent the creation of a product view. You can also set this preference with the **Product View Creation Preferences→View Toggles to consider** option.

Enable late loading of product views

Users can select multiple product views in the product view gallery and send them to the Lifecycle Viewer or stand-alone Lifecycle Visualization. The product views open one at a time and any configuration of the original structure is retained.

As an administrator, you can enable late loading of product views, by using the **Vis_PV_AllowLateLoading** site preference. Late loading is useful to reduce the time it takes to load a large number of product views at once in stand-alone Lifecycle Visualization. A late loaded product view appears as a thumbnail on the 3D Snapshots page, but the entire product view is not loaded until you fully open the product view by applying the 3D snapshot.

When late loading is enabled, you can set how partially loaded product views are handled when you save a session file or a PLMXML file. You do this by setting the **Vis_PV_LateLoadSaveOp** site preference.

Setting assembly preferences

Setting assembly preferences

The Teamcenter administrator determines the following functionality by setting the relevant preferences.

TCVIS_allow_NGC_expand_mode preference

When the data panel viewer is open, and a structure line has an attached occurrence note of type **UG Geometry** , you can control the expansion of the structure by setting the **TCVIS_allow_NGC_expand_mode** preference:

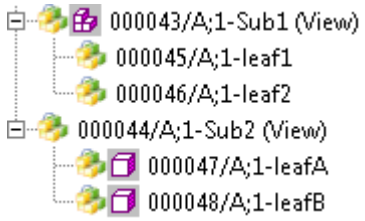
Value	Description
0	The structure is not expanded.
1	The structure is expanded to show the geometry.
2	The structure is expanded but does not show the geometry.

This preference has no effect on the Teamcenter Lifecycle Viewer or stand-alone Teamcenter lifecycle visualization.

TCVIS_enable_NGC_feature preference

The **TCVIS_enable_NGC_feature** preference controls how non UG geometry associated with occurrence nodes displays. When the **TCVIS_enable_NGC_feature** preference is set to **True**, the **TCVIS_allow_NGC_expand** preference is set to **True**, and the occurrence node has the UG Geometry value set to **No**, the structure displays as follows:

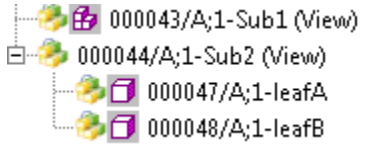
- If the occurrence node is an assembly, it is preceded by a icon. If the occurrence node is a leaf, it is preceded by a icon.
- If the occurrence node is an assembly, it can be expanded to show the children.



- If the graphics viewer is open in the data panel, it displays the occurrence node.

However, if the **TCVIS_allow_NGC_expand_mode** preference is set to **False**, the structure displays as follows:

- If the occurrence node is an assembly, it is preceded by a icon. If the occurrence node is a leaf, it is preceded by a icon.
- If the occurrence node is an assembly, it cannot be expanded to show the children.



- If the graphics viewer is open in the data panel, it does not display the occurrence node.

When the **TCVIS_enable_NGC_feature** preference is set to **False**, any features related to non UG geometry are ignored, consequently the **TCVIS_allow_NGC_expand** preference is also ignored. Any occurrence node (even those with the **UG Geometry** property set to **No**) are treated as normal BOM line occurrences. The non UG geometry icon does not display and the node may be expanded in the graphics viewer.

If the **TCVIS_enable_NGC_feature** preference is not set, the system uses the default value of **True**.

If this preference is set to **False**, the value of the **TCVIS_allow_NGC_expand** preference is ignored.

This preference has no effect on the Teamcenter Lifecycle Viewer or stand-alone Teamcenter lifecycle visualization.

Setting assembly level of detail

You can display a part or assembly in the embedded viewer by selecting the corresponding line in the assembly tree and clicking the **Graphics** tab. If a JT file is associated with the line (that is, if the item revision corresponding to the line has a **DirectModel** dataset containing a JT file) Teamcenter displays the JT file in the viewer. Any descendents of that line with attached JT files are displayed likewise.

By defining a naming convention for reference sets and JT files, you control the behavior of assemblies in the viewer. You specify the naming convention by setting the **JT_File_OverrideChildren_Refsets** preference. The default setting of this preference is **unset**, but you can set it to any NX reference set names, for example, **SIMPLE** or **FULL**.

If you click on a structure line, Teamcenter displays the corresponding JT file. If there is no JT file, the viewer shows all of its immediate children. Teamcenter repeats this process recursively down each branch of the product structure until it finds and displays a JT file. However, if the JT file is in the preference list, none of its children are displayed.

If you do not set the **JT_File_OverrideChildren_Refsets** preference, Teamcenter continues processing and displaying the children.

Determining JT file priorities

Each structure element typically has a maximum of one JT file in its associated **DirectModel** dataset. Teamcenter always displays this file in the viewer. However, a **DirectModel** dataset can contain several JT files with different names. Therefore, you must set rules that determine the file to be displayed. By default, Teamcenter uses the JT file that matches the NX **REF SET** occurrence note for the structure element. That is, Teamcenter tries to use the same reference set as the Teamcenter Integration for NX part file. You can override the default behavior by specifying the order in which the reference sets are loaded. To do that, you use the **JT_File_Priority_Refsets** preference.

For more information about this preference, see *Administration Data Documentation* on Support Center.

Determining excluded reference sets

You can set the **JT_File_Excluded_Refsets** preference to define reference set names that are not visualized in an assembly. However, the reference sets you exclude with this preference can be viewed, if required, using the **Replace JT** option. This preference does not have a default setting and you can set it to any of the following values:

- **Empty**

  This reference set name is appropriate for most sites. NX uses this name to identify geometry that should not be visualized in an assembly.

- Entries other than **Empty**

  Sites that define reference set names to identify geometry that should not be visualized can add the names to the preference instead of **Empty**.

29. Configure BOM grading on the rich client

BOM grading is a validation engine for bills of materials (BOMs). It allows customers to validate their product's BOM and ensure all parts meet configurable criteria like approved parts, not obsolete, and compliant RoHS. This can be difficult if done manually because a BOM can contain thousands of parts. Within the Business Modeler IDE, a user can create conditions and associate them with business objects using verification rules.

In addition, parts approved for use on one project may not be approved for use on another project. For example, a part can pass a European compliance standard but not a U.S. one. This means a BOM potentially could be graded against multiple sets of conditions depending on the product context (target market, manufacturing plant, and so on).

The Validation Manager application allows user-defined contexts (sets of conditions) by creating checkers.

The **Tools→BOM Grading** menu command in Structure Manager performs a BOM validation against selected checkers. The BOM grading results viewer allows users to review, analyze, filter, and override validation results.

To configure BOM grading objects in the Business Modeler IDE:

1. If you have not already done so, create a custom template project to hold your data model changes.

2. Create conditions.

   You must create conditions to evaluate parts to ensure they are valid in BOMs. To create a condition, open the **Extensions\Rules** folders, right-click the **Conditions** folder, and choose **New Condition**.

   BOM grading supports the following condition signatures:

   ```
   (WorkspaceObject r)
   (WorkspaceObject r, BOMLine o)
   ```

   For example, if you want to check if a part has a released status or not using BOM grading, set a condition similar to the following that uses the **INLIST** function:

   ```
   TestReleaseStatusCondition (ItemRevision o) =:
   INLIST("Obsolete",o.release_status_list,"name")
   ```

3. Assign the conditions to part types using **Teamcenter Component** objects and verification rules.

Create a **Teamcenter Component** object to link conditions to business objects, and create verification rules to set the scope. To create a **Teamcenter Component** object, in the **Extensions** folder, right-click the **Teamcenter Component** folder and choose **New Teamcenter Component**. To create a verification rule, on the menu bar, choose **BMIDE→Editors→Verification Rules Editor**, and click the **Add** button to the right of the **Verification Rule** table.

The following COTS objects are used in BOM grading:

- The **Fnd0BOMGrading Teamcenter Component** object is provided in the **Extensions\Teamcenter Component** folder.
- The **Fnd0BOMGrading** list of values (LOV) located in the **Extensions\LOV** folder.

4. Deploy the template to the server for use in BOM grading.