TEAMCENTER

Tc XML and PLM XML Configuration for Data Import and Export

Teamcenter 2312

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

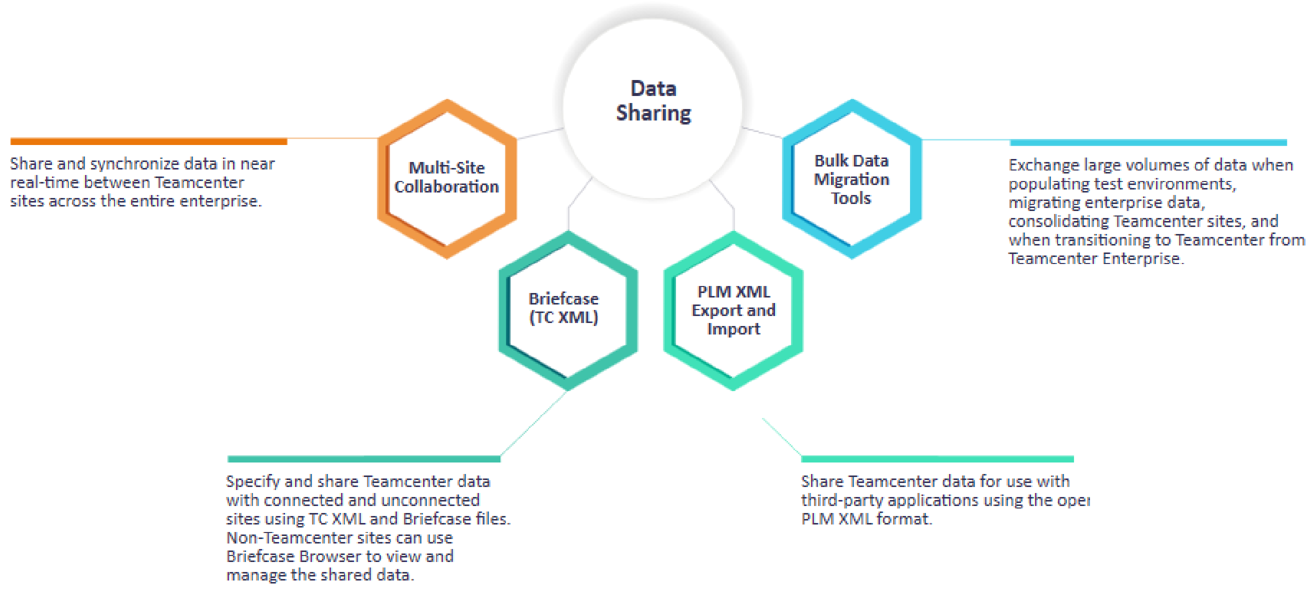
1. Getting started with PLM XML/TC XML Export Import Administration

What is PLM XML/TC XML Export Import Administration?

When sharing data with other organizations or for use with third-party applications, you want to tightly control the scope, versions, and formats of the data being shared. Use the PLM XML/TC XML Export Import Administration application to create transfer mode objects that contain rules that configure import or export operations. Doing so simplifies the import and export process for end users, who can easily select a transfer mode object when importing or exporting data without needing to know the specific rules the transfer mode contains.

You do not actually import or export data using PLM XML/TC XML Export Import Administration; you use other applications, such as My Teamcenter or Structure Manager.

Sharing TC XML and sharing PLM XML are two of several solutions Teamcenter offers to meet different data sharing needs.


Data Sharing
Multi-Site Collaboration
Share and synchronize data in near real-time between Teamcenter sites across the entire enterprise.
Bulk Data Migration Tools
Exchange large volumes of data when populating test environments, migrating enterprise data, consolidating Teamcenter sites, and when transitioning to Teamcenter from Teamcenter Enterprise.
Briefcase (TC XML)
Specify and share Teamcenter data with connected and unconnected sites using TC XML and Briefcase files. Non-Teamcenter sites can use Briefcase Browser to view and manage the shared data.
PLM XML Export and Import
Share Teamcenter data for use with third-party applications using the open PLM XML format.

Where do I go from here?

Business User	
Exchanging data as PLM XML.	See Using Teamcenter to export and import data with PLM XML.
Exchanging data using TC XML.	See Data sharing concepts.

Importing objects modified since a previous import.	See **Delta import overview**.
Administrator	
Learn more about transfer mode objects.	See **Transfer mode objects**.
Learn more about transfer option sets.	See **What are transfer option sets**.
Customizing PLM XML import and export extensions.	See **Available customization points**.
Solving import and export issues.	See **Troubleshooting PLM XML exports and imports**.
Exchanging data as PLM XML.	See **Exporting and importing using PLM XML**.

Requirements for using PLM XML/TC XML Export Import Administration

Prerequisites

To use PLM XML/TC XML Export Import Administration, you should know:

- Specific Teamcenter applications.

  You must understand the specific application you intend to import or export data from. For example, My Teamcenter, Structure Manager, and Manufacturing Process Planner are three of the applications that use PLM XML. Each of these applications manages and transforms different Teamcenter data for different purposes.

- The Teamcenter data model.

  You must understand the Teamcenter data model, including the overall data that is stored. For example, some of the data stored are items, item revisions, and forms, along with the constructs holding the data such as classes, attributes, types, properties, and relations. Each application manages and transforms data within these representations. To see class, attribute, and type information, use the Business Modeler IDE.

- Persistent and run-time objects.

  You should know the difference between persistent data stored in the database and objects that are created through the application layer at run time.

- The PLM XML schema.

	You must understand the PLM XML industry standard schema. This allows you to understand the XML-formatted neutral representation and organization of the data for import and export to Teamcenter.
Enable PLM XML/TC XML Export Import Administration	PLM XML/TC XML Export Import Administration does not need to be enabled before you use it. If you have trouble accessing PLM XML/TC XML Export Import Administration, see your system administrator; it may be a licensing issue. Note: You can log on to Teamcenter only once. If you try to log on to more than one workstation at a time, you see an error message.
Configure PLM XML/TC XML Export Import Administration	PLM XML/TC XML Export Import Administration does not need to be configured.
Start PLM XML/TC XML Export Import Administration	Click **PLM XML/TC XML Export Import Administration** in the navigation pane.

Basic concepts for using PLM XML

PLM XML is a data (and metadata) representation for the exchange of product life cycle information. It is a W3C standard supported by multiple XML schemas. The PLM XML schemas and documentation enable you to introduce PLM XML support into your applications. Data represented in the schemas include product structure, geometry, visualization, features, application associativity, data ownership, and delta (change). Several dominant schemas include:

- Base

  Core PLM XML schema markup

- MRO

  In-service product data

- Delta

  A change to be applied to a PLM XML file

- Annotation

  PMI, GD&T, and markup

- PDM

  Product data management (PDM) schema (workflow, revision rule, form, folder)

- MPM

  Manufacturing process management

The PLM XML schemas are available within your Teamcenter installation at *TC_ROOT\reference_files\PLMXML\schemas*. The schemas can be downloaded freely for development of collaborative interoperable applications.

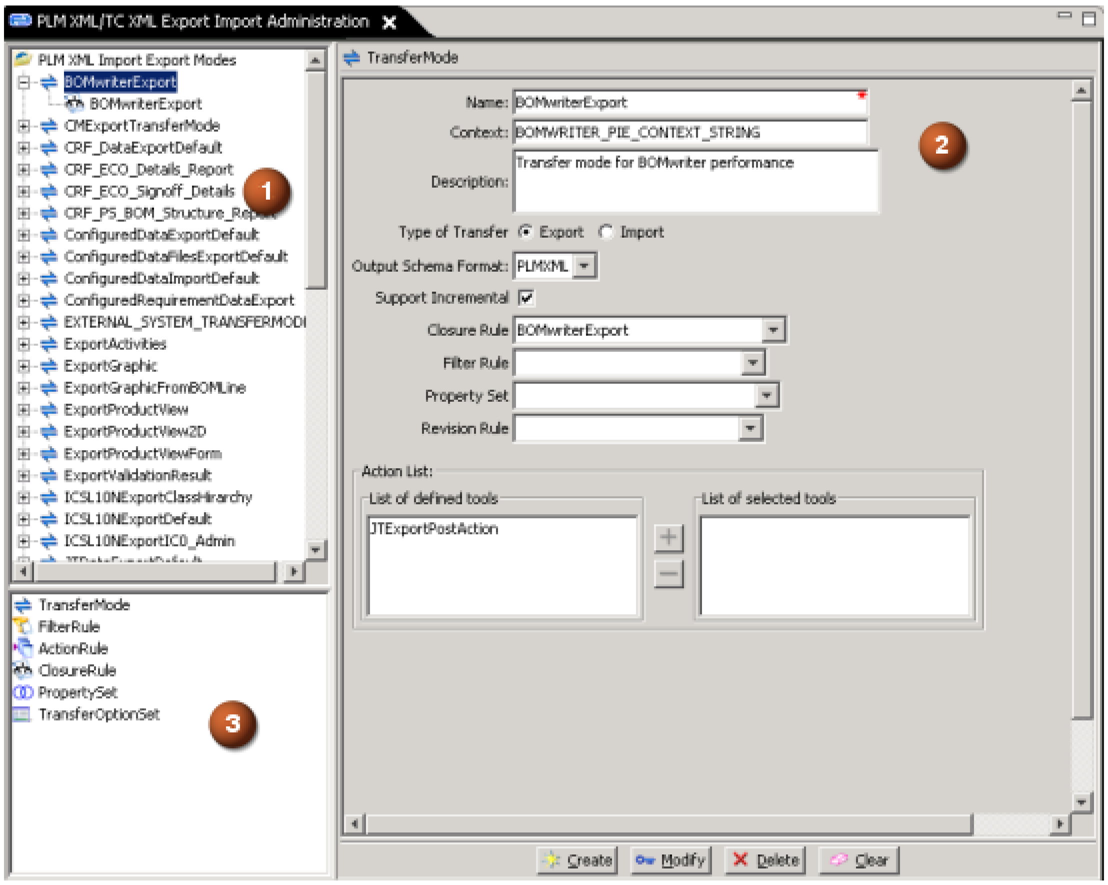
The URIs that appear in the headers of PLM XML and TC XML files serve as namespace names, which are unique identifiers for an XML vocabulary. Although they are URIs, they are not used to identify and retrieve Web addresses. For example:

```
<?xml version=”1.0” encoding=”utf-8”?>
<PLMXML xmlns=”http://www.plmxml.org/Schemas/PLMXMLSchema”
```

This practice is consistent with **W3C standards for namespaces**.

PLM XML/TC XML Export Import Administration in the rich client

Access PLM XML/TC XML Export Import Administration using the rich client. All PLM XML/TC XML Export Import Administration menus are standard Teamcenter rich client menus. All PLM XML/TC XML Export Import Administration buttons are standard Teamcenter interface buttons.



TransferMode

Name: BOMwriterExport

Context: BOMWRITER_PIE_CONTEXT_STRING

Description: Transfer mode for BOMwriter performance

Type of Transfer [✓ Export] Import

Output Schema Format: PLMXML

Support Incremental ✓

Closure Rule BOMwriterExport

Filter Rule

Property Set

Revision Rule

Action List:

List of defined tools: JTExportPostAction

List of selected tools:

1	Object tree	Lists the transfer modes and associated objects. When selected, the details for the object are displayed in the **TransferMode** pane.
2	TransferMode pane	Displays the details of the selected object. You can create, modify, and delete objects in this pane.
3	TransferMode tree	Lists the objects that you can add to a transfer mode.

Basic tasks using PLM XML/TC XML Export Import Administration

With PLM XML/TC XML Export Import Administration, you create transfer mode objects that help end users export and import Teamcenter objects and system data. To create a transfer mode object, you may need to:

- Create or edit a closure rule.
- Create or edit a filter rule.
- Create or edit an action rule.

- **Create or edit a property set.**

Once you create these items, you can **assemble them into a transfer mode object**.

You can also **create a transfer option set** for use when exchanging Teamcenter data.

Using the TC XML schema

TC XML is a PLM XML schema especially designed to work with the Teamcenter data exchange tools. You use TC XML and transfer option sets with Teamcenter data exchange tools to move data between Teamcenter and Teamcenter Enterprise sites.

Syntax conventions used in this guide

This guide uses a set of conventions to define the syntax of Teamcenter commands, functions, and properties. Following is a sample syntax format:

> **harvester_jt.pl** [*bookmark-file-name bookmark-file-name* ...]
> [*directory-name directory-name* ...]

The conventions are:

Bold	Bold text represents words and symbols you must type exactly as shown. In the preceding example, you type **harvester_jt.pl** exactly as shown.
Italic	Italic text represents values that you supply. In the preceding example, you supply values for *bookmark-file-name* and *directory-name*.
text-text	A hyphen separates two words that describe a single value. In the preceding example, *bookmark-file-name* is a single value.
[ ]	Brackets represent optional elements.
...	An ellipsis indicates that you can repeat the preceding element.

Following are examples of correct syntax for the **harvester_jt.pl** command:

```
harvester_jt.pl
harvester_jt.pl assembly123.bkm
harvester_jt.pl assembly123.bkm assembly124.bkm assembly125.bkm
harvester_jt.pl AssemblyBookmarks
```

2. Working with transfer mode objects

Transfer mode objects

Transfer mode objects are created in PLM XML/TC XML Export Import Administration and are displayed as options when users import or export objects or system data using one of the Teamcenter applications. These objects allow users to export and import data with minimal information other than the name of the transfer mode object that they should use. Examples include **ToCompanyA**, **FromCompanyB**, or **ToCompanyB**.

Note:

When importing data into Teamcenter, object names and IDs that exceed their maximum field length are truncated.

Transfer mode objects are made up of the following items that configure the import or export operation.

- **Closure rules**

  Defines the scope of the data translation.

- **Filter rules**

  Use conditions to apply a finer level of control over which data gets translated along with the primary objects.

- **Action rules**

  Defines the sets of methods that can be called before, during, and after the translation.

- **Property sets**

  Provides a mechanism for PLM XML objects to have arbitrary properties in the form of **UserData** elements.

- **Context**

  Provides a mechanism for using method extensions to export or import a specific Teamcenter business object or PLM XML element.

  When defining a transfer mode, you can leave the **Context** box blank.

- **Revision Rule**

Specifies a transfer mode revision rule, though the rule is not applied to the target data during export and import.

Managing closure rules

Introduction to closure rules

Note:

You must understand the Teamcenter class and attributes structure (the POM schema) when you create closure rules. Administrators can view the POM schema in the Business Modeler IDE **Classes** view or by right-clicking a class in the **Classes** view and choosing **Open in UML Editor**.

Closure rules control the scope of the data translation on both input and output. They specify how the data structure is traversed by specifying which relationships are of interest and what should be done when these relationships are encountered.

Closure rules are comprised of five parts: primary object selector, secondary object selector, relation selector, action, and an optional conditional clause. Closure rules are generally grouped. The closure rule clauses are evaluated in order for each target object. The first applicable rule determines the disposition of the related object.

A rule is applicable only if the target object fits the primary selector, the secondary object fits the secondary selector, the relationship between the two objects fits the relationship selector, and the conditional clause evaluates to **TRUE**. The action of the applicable closure rule clause determines how the translator handles the secondary object.

If no rule applies, the object is skipped. If the action is **TRAVERSE**, the object becomes another target of the translation. If the action is **PROCESS**, the secondary object is written (into either PLM XML or the Teamcenter database), but no closure rule clauses are applied to the secondary object's related objects. If the action is **SKIP**, the object remains as is.

Use PLM XML/TC XML Export Import Administration to create, edit, and delete closure rules.

The action type on closure rule clauses may appear as **TRAVERSE_AND_PROCESS**, **PROCESS+TRAVERSE**, or **Traverse_and_Process**. All instances are treated as **TRAVERSE_AND_PROCESS**.

Syntax for closure rules

The following example illustrates the syntax for closure rules:

{CLASS | TYPE}.*primary-class-or-type-name***:{CLASS | TYPE}.***secondary-class-or-type-name***:**
{ATTRIBUTE | PROPERTY | RELATIONS2P | RELATIONP2S | REFBY | OCCTYPE | CONTENT | FUNCTION}.*name-of-connecting-relationship***:{***directive1***+***directive2***+***directive3***}:** *conditional-clause*

Caution:

Use caution when using an asterisk (*****) wildcard. Using an asterisk, rather than specifying a name after the primary or secondary object keyword, specifies that all classes or types fit the primary or secondary object criteria.

The following closure rule attempts to translate every object in the database:

```
CLASS.*:CLASS.*:PROPERTY.*:Traverse_and_Process
```

The qualifier for the primary object selector or secondary object selector can be **CLASS** or **TYPE**:

- If **TYPE.***type_name* is used, the type of the object must exactly match the type name specified in the clause. For example, **TYPE.DirectModel** fits only the **DirectModel**-type datasets.
- If **CLASS.***class_name* is used, the class of the object must be the same as the class name specified in the clause or a descendent in the class hierarchy. For example, **CLASS.Dataset** fits any instance of **Dataset** or any instance of its subclasses.

Clause item	Definition
Primary object selector	Specifies the primary object class or type to be translated. The class or type name is preceded by the **CLASS** or **TYPE** keyword and a period (**.**). For example: `CLASS.Item`
Secondary object selector	Specifies the secondary object class or type to be translated. The class or type name is preceded by the **CLASS** or **TYPE** keyword and a period (**.**). For example: `TYPE.Item`
Relation selector	Specifies the relationship between the secondary and primary object class or type. The relation name is preceded by a keyword and a period. For example: `ATTRIBUTE.items_tag` Using a wildcard (*****) rather than specifying a name after any of the keywords indicates that all relations of this type fit the rule. The following keywords apply to relation selectors: **PROPERTY** Specifies the name of the Teamcenter property of the primary object that refers to the secondary object.

Clause item	Definition
	ATTRIBUTE Specifies the name of the Teamcenter attribute of the primary object that refers to the secondary object. **RELATIONS2P** Specifies the relation type that binds the primary and secondary object and indicates that the traversal begins at the secondary object. **RELATIONP2S** Specifies the relation type that binds the primary and secondary object and indicates that the traversal begins at the primary object. **REFBY** Specifies the name of the attribute of the secondary object that relates the secondary object to the primary object. **OCCTYPE** Specifies the type of occurrence in a BOM window that relates the primary object to the secondary object. **CONTENT** Specifies that the relation being traversed is in a PLM XML→Teamcenter translation and that the secondary object is understood to be an element contained in the primary object. The name is ignored for content. **FUNCTION** Specifies that the named function be executed. Note: The **FUNCTION** relation selector applies only to TC XML data exchange use cases. It has no effect when used in closure rules for PLM XML export and import.

Clause item	Definition

	When **FUNCTION** is used as the relationship selector, you must use the conditional clause to specify the function arguments in the form of **FUNC_ARGS**. If the function has more than one argument, they must be separated by semicolons (for example, **FUNC_ARGS(***arg1***;***arg2***;***arg3***)**). An argument can be one of the following forms:
	
	• **$option_symbol_name**
	
	The value of this option is passed to **FUNCTION**.
	
	• **“abc”**
	
	The value **abc** is passed to **FUNCTION**, similar to a literal in a programming language.
	
	• **PRIMARY.property_name**
	
	The value of **property_name** on **primaryObject** is passed to **FUNCTION**. **SECONDARY** and **RELATION** are not allowed as they are not yet available while processing **FUNCTION**.
	
	For example, a closure rule clause that uses **FUNCTION** appears as follows:
	
	`CLASS.Item:CLASS.ItemRevision:FUNCTION.`
	`   process_revision_selector:`
	`   TRAVERSE_AND_PROCESS:`
	`   FUNC_ARGS($opt_rev_select)`
	
	This clause specifies that during traversal, when the transfer mode encounters an instance of **Item** as a primary object, it must determine the necessary **ItemRevision** secondary objects and execute the appropriate **FUNCTION** registered against the **process_revision_selector** function handle.
	
	The possible revision rule selector values for **$opt_rev_select** are:
	
	• **allItemRevisions**
	
	(**Include All Revisions** in Multi-Site Collaboration)
	
	• **latestRevisionOnly**
	
	(**Latest Revision Only** in Multi-Site Collaboration)

Clause item	Definition
	• **latestWorkingRevisionOnly** (**Latest Working Revision Only** in Multi-Site Collaboration) • **latestWorkingAnyOnly** (**Latest Working/Any Release Status** in Multi-Site Collaboration) • **latestReleasedRevisionOnly** (**Latest Any Release Status** in Multi-Site Collaboration) • **selectedRevisionsOnly** (**Selected Revision(s) Only** in Multi-Site Collaboration) • **specificStatusOnly_***status-name* (**Specific Release Status Only** in Multi-Site Collaboration)
Action Type	Specifies the traversal actions to be taken. It must be one or a combination of the following keywords:

Tip:

Define action combinations by using the shift key or manually typing the action, for example:

```
TRAVERSE_AND_PROCESS
```

SKIP

Specifies that the relationship should not be followed. This directive is useful to eliminate special cases before a more general rule is reached. For example, **Document** is a sub-class of **Item**. With the **SKIP** clause, secondary objects with the **Document** class are skipped and any other **Item** and its sub-classes are processed for the specified GRM **EC_reference_item_rel**.

```
CLASS.Item:CLASS.Document:P2S: EC_reference_item_rel:SKIP
CLASS.Item.CLASS.Item:P2S: EC_reference_item_rel:PROCESS
```

The **SKIP** action cannot be combined with any other action.

SKIP_GRM

Clause item	Definition

	(For TC XML export only) Specifies that the relation must not be exported. For example, if the following closure rule clause is used, the relation **IMAN_UG_wave_position** between **WorkspaceObject** and **WorkSpaceObject** is not exported.
	
	`CLASS.WorkspaceObject:CLASS.WorkspaceObject.`
	`RELATIONP2S.IMAN_UG_wave_position:SKIP_GRM`
	
	For TC XML exports, the **SKIP_GRM** action differs from the **SKIP** action:
	
	• **SKIP**
	
	The primary object and GRM are exported as full objects, and the secondary object is exported as a stub.
	
	• **SKIP_GRM**
	
	The primary object is exported as a full object, and the GRM and secondary objects are ignored (neither processed nor traversed).
	
	In the following closure rule example, the relation **IMAN_UG_wave_position** and the primary workspace object are exported as full objects, but the secondary object is exported as a **POM_stub** object.
	
	`CLASS.WorkspaceObject:CLASS.WorkspaceObject.`
	`   RELATIONP2S.IMAN_UG_wave_position:SKIP`
	
	The **SKIP_GRM** action cannot be combined with any other action.
	
	PROCESS
	
	Processes the secondary object but does not traverse further, unless combined with the **TRAVERSE** action.
	
	TRAVERSE
	
	Specifies that the object becomes another target of the translation.
	
	Note:
	
	The **PROCESS** and **TRAVERSE** actions can be combined by using the shift key.

Clause item	Definition
	REFERENCE Writes a PLM XML pointer entity. **CHANGE_CLOSURE_RULE** Alters the closure rules in force when an entity being processed contains its own closure rule element. **DO** Processes the secondary object and specifies that the object becomes another target of the translation. Note: The **PROCESS** and **REFERENCE** actions are mutually exclusive.
Conditional Clause	Evaluates to **TRUE** or **FALSE** and is expressed in the following form: *Condition := Expression1 == Expression2* *Condition := Expression1 != Expression2* *Condition := (Condition) && (Condition)* *Condition := (Condition) || (Condition)* *Condition := (!Condition)* *Expression1* can be: • **PRIMARY.property_name**, **SECONDARY.property_name**, or **RELATION.property_name** Evaluates the value of the **property_name** property on the primary object, secondary object, or relationship object as appropriate. • **$(option_name)** **option_name** is a symbol set on the traversal engine. This allows the rule to be in effect if a program state is set, which then allows the closure rule to take advantage of internal program states for particularly complex traversal operations. *Expression2* can be a string literal enclosed in double quotation marks (for example, **“value1”**).

Clause item	Definition
	In all cases, the expressions are converted to character strings, a string compare is performed, and the clause is evaluated. The closure rule qualifies only if the conditional clause evaluates to **TRUE**. Conditional clauses must specify the exact property values to make a proper comparison; wild card characters are not allowed in the comparison.
Conditional Clause	Following are examples of conditions: • **PRIMARY.object_name == "MyObject"** Evaluates the current closure rule clause only when the **object_name** property on the **PRIMARY** object is set to **MyObject**. • **$opt_jt_ds == "true"** Evaluates the current closure rule clause only when the Traversal engine has the **opt_jt_ds** option set to **true**. • **PRIMARY.object_name == "MyObject" && $opt_jt_ds == "true"** • **TYPE.CfgAttachmentLine : TYPE.CfgAttachmentLine:PROPERTY. me_cl_child_lines: TRAVERSE_AND_PROCESS:SECONDARY.al_source_class == "Dataset"** • **SECONDARY.object_type != "MyType"** Following are examples of invalid conditions: • **PRIMARY.object_count > 3** • **SECONDARY.object_id = "233*"** • **PRIMARY.object_name = "TEST%"** In some cases, the conditional clause may not be sufficient to achieve the desired result. For example, to exclude any **BOMLine** with a certain prefix in the **object_name**, a conditional clause cannot be formulated.

Create closure rules

1. Choose the **ClosureRule** node in the **TransferMode** tree (located in the lower-left pane of the PLM XML/TC XML Export Import Administration window).

The system displays the **ClosureRule** pane.

2. Type a name for the rule in the **Traversal Rule Name** box.

   Do not begin your closure rule name with **FT_** or **__**. These prefixes are reserved for internal use only.

3. Optionally, type a description of the closure rule in the **Description** box.

4. In **Scope of Traversal**, click either **Export** or **Import**.

   Use this property only in defining closure rules. The **BOMLine** property **bl_attachments** must not be used by Teamcenter applications because it returns run-time objects that do not persist and are removed after the call to the property.

   Alternatively, Siemens Digital Industries Software recommends using the following *Teamcenter Services* (SOA) APIs to get or close attachment lines:

   manufacturing::StructureManagment::getBOMLineAttachments

   manufacturing::StructureManagment::closeAttachmentWindow

5. In the **Output Schema Format** menu, choose either **PLMXML** for a standard PLM XML import or export or **TC XML** if you are using other Teamcenter data sharing methods to move data between Teamcenter and Teamcenter Enterprise sites.

6. Create the ordered clauses that specify how the data structure is traversed.

   a. Click the **Add clause** button + to the right of the clause table.

      The system displays a blank row in the table.

   b. Select one of the following keywords from the **Primary Object Class Type** list:

      CLASS

      TYPE (available only if **Scope of Traversal** is **Export**)

   c. Type the primary object's name in the **Primary Object** box.

   d. Select one of the following options from the **Secondary Object Class Type** list:

      CLASS

      TYPE (available only if **Scope of Traversal** is **Export**)

e. Type the secondary object's name in the **Secondary Object** box.

f. Select one of the **Relation Type** options listed in the **Syntax for closure rules** table.

g. Type the name of a single property (or multiple properties) or objects in the **Related Property Or Object** box.

h. Select one or a combination of the following options from the **Action Type** list:

   SKIP

   Specifies that the relationship should not be followed. This directive is useful to eliminate special cases before a more general rule is reached. For example, you could use the clauses in the following example to get all occurrence notes from a BOM line except manufacturing notes:

```
TYPE.BOMLine:TYPE.MfgNote.
   PROPERTY.bl_all_notes:SKIP
TYPE.BOMLine:TYPE.*:PROPERTY.
   bl_all_notes:PROCESS
```

   The **bl_all_notes** property points only to notes; therefore, the relaxed secondary condition of the second clause allows all note types to be included. The manufacturing notes are skipped because the clauses are ordered.

   The **SKIP** action cannot be combined with any other action.

   PROCESS

   Processes the secondary object but does not traverse further, unless combined with the **TRAVERSE** action.

   TRAVERSE

   Specifies that the object becomes another target of the translation.

   The **TRAVERSE** and **PROCESS** actions can be combined by using the shift key.

   REFERENCE

   Writes a PLM XML pointer entity.

   CHANGE_CLOSURE_RULE

   Alters the closure rules in force when an entity being processed contains its own closure rule element.

The **PROCESS** and **REFERENCE** actions are mutually exclusive.

i. Optionally, type a conditional clause in the **Conditional Clause** box.

j. Repeat the previous steps to add additional clauses to the closure rule.

k. Optionally, change the precedence of the rule clauses by selecting a row in the table and using the **Move Up** ▲ or **Move Down** ▼ buttons to the right of the clause table.

7. Click the **Create** button.

   The system displays the new rule in the **ClosureRule** node of the **TransferMode** tree. This rule can now be used when building transfer mode objects.

Edit closure rules

You can edit closure rules by adding, removing, or changing the definition of the clauses that make up the rule. You can also change the precedence of the clauses.

1. Select the node in the **ClosureRule** branch of the **TransferMode** tree that corresponds to the rule you want to modify.

2. Modify the contents or precedence of the clauses, as described in *Create closure rules*.

3. Click the **Modify** button.

   The system saves the changes to the database.

Delete closure rules

1. Select the node in the **ClosureRule** branch of the **TransferMode** tree that corresponds to the rule you want to delete.

2. Click the **Delete** button.

   The system displays the **Delete Confirmation** dialog box.

3. Click the **Yes** button to delete the rule or click the **No** button to cancel.

Map attributes with a closure rule

To communicate an object's attribute information between an application and Teamcenter during open, export, save, or import operations, you can map those attributes with PLM XML.

The application object is stored as an item and item revision, and the associated application data file is a named reference of the dataset that is created as part of the item revision. You can map the application attributes to any property that can be defined in a mapping list. For example, it can be one of the properties in the item revision primary (master) form or the item primary form.

To map attributes, you must write a closure rule with the following clause and set it in the transfer mode used for export:

Clause item	Value
Primary Object Class Type	**CLASS**
Primary Object	**Dataset**
Secondary Object Class Type	**CLASS**
Secondary Object	*****
Relation Type	**PROPERTY**
Related Property Or Object	**AppAttr**
Action Type	**PROCESS**

Managing filter rules

What are filter rules?

Filter rules allow a finer level of control over the data that gets translated along with the primary objects by specifying that a user-written function is called to determine the operation applied against a given object. Filter rules conform to the syntax described in *Syntax for closure rules*.

Closure rules only allow you to control what is going to happen when moving from one object to the next. But if you base your decision on more complex criteria, you need a filter rule. For example, you need a filter rule if you want the PLM XML import export to follow these steps: if the item revision has a **UGMASTER** dataset, translate it and skip all other datasets; if there is no **UGMASTER** dataset, export the JT dataset. Closure rules cannot base their traversal on what child element is in existence, but filter rules can.

Use PLM XML/TC XML Export Import Administration to create, edit, and delete filter rules.

Create filter rules

1. Choose the **FilterRule** node in the **TransferMode** tree (located in the lower-left pane of the PLM XML/TC XML Export Import Administration window).

   The system displays the **FilterRule** pane.

2. Type a name for the rule in the **Filter Rule Name** box.

3. Optionally, type a description of the rule in the **Description** box.

4. In **Scope of Filter**, click either **Export** or **Import**.

5. From the **Output Schema Format** menu, choose either **PLMXML** for a standard PLM XML import or export or **TC XML** if you are using other Teamcenter data sharing methods to move data between Teamcenter and Teamcenter Enterprise sites.

6. Create the ordered clauses that specify how the data structure will be traversed:

   a. Click the **Add clause** button + to the right of the clause table.

   b. Select one of the following keywords from the **Object Class Type** list:

      CLASS

      TYPE (available only if **Scope of Filter** is **Export**)

   c. Type the object's name in the **Object Name** box.

   d. Select a rule in the **Filter Rule Name** list.

7. Click the **Create** button.

   The system displays the new rule in the **FilterRule** node of the **TransferMode** tree. This rule can now be used when building transfer mode objects.

Edit filter rules

You can edit filter rules by adding, removing, or changing the definition of the clauses that make up the rule. You can also change the precedence of the clauses.

1. Select the node in the **FilterRule** branch that corresponds to the rule you want to modify.

2. Modify the contents or precedence of the clauses, as described in *Create filter rules*.

3. Click the **Modify** button.

   The system saves the changes to the database.

Delete filter rules

1. Select the node in the **FilterRule** branch that corresponds to the rule you want to modify.

2. Click the **Delete** button.

   The system displays the **Delete Confirmation** dialog box.

3. Click the **Yes** button to delete the rule, or click the **No** button to cancel.

Managing action rules

What are action rules?

Action rules are sets of methods that can be called before, during, and after the translation. The Teamcenter PLM XML import export framework allows for the pre-functional, during-action, and post-functional processing of action rules on behalf of the user. For example:

- **Pre-action**

  These rules are executed before the translation occurs. You can use these to verify or prepare the application, data, or Teamcenter for the translation.

- **During-action**

  When the PLM XML import export is iterating through the data in your translation, the translation engine verifies if you have during-action rules after all objects are translated but before applying any style sheets (**.xslt**) in the case of an export. If so, the rules are executed and the **.xslt** transformations are applied.

- **Post-action**

  Post-processing is done after the translation is complete, but the session still has control over the translation. With post-action rules, you can verify the specific errors or content translations of each object.

Use PLM XML/TC XML Export Import Administration to create, edit, and delete action rules.

Create action rules

1. Choose the **ActionRule** node in the **TransferMode** tree (located in the lower-left pane of the PLM XML/TC XML Export Import Administration window).

   The system displays the **ActionRule** pane.

2. Type a name for the rule in the **Action Rule Name** box.

3. Optionally, type a description of the action rule in the **Description** box.

4. In **Scope of Action**, click either **Export** or **Import**.

5. From the **Output Schema Format** menu, choose either **PLMXML** for a standard PLM XML import or export or **TC XML** if you are using other Teamcenter data sharing methods to move data between Teamcenter and Teamcenter Enterprise sites.

6. In the **Location of Action** list, select when you want the action handler to run **Pre Action**, **During Action**, or **Post Action**. When the handler executes depends if you are importing or exporting.

Option	Scope of action =Teamcenter (export)	Scope of action = PLMXML (import)
Pre Action	Before opening the PLM XML file	Before the XSLT file is imported
During Action	After translation	Before translation
Post Action	After the XSLT file is exported	After the PLM XML file is closed

7. In the **Action Handler** list, select the action handler you want executed.

8. Click the **Create** button.

   The system displays the new rule in the **ActionRule** node of the **TransferMode** tree. This rule can now be used when building transfer mode objects.

Edit action rules

You can edit action rules by changing the action handler, scope of action, or location of action.

1. Select the node in the **ActionRule** branch of the **TransferMode** tree that corresponds to the rule you want to modify.

2. Modify the contents or precedence of the clauses, as described in *Create action rules*.

3. Click the **Modify** button.

   The system saves the changes to the database.

Delete action rules

1. Select the node in the **ActionRule** branch of the **TransferMode** tree that corresponds to the rule you want to delete.

2. Click the **Delete** button.

The system displays the **Delete Confirmation** dialog box.

3. Click the **Yes** button to delete the rule or click the **No** button to cancel.

Managing property sets

Property sets

Property sets provide a nonprogrammatic way to control what is placed in the **UserData** element. The **UserData** element is a container for a list of name-value pairs that allows any attribute or property in an **ImanObject** object that is not in the PLM XML equivalent to be stored.

The Teamcenter data model is richer than the PLM XML model and you can extend the Teamcenter data model further. Property sets are a way to get Teamcenter data into the PLM XML file that may not be in the PLM XML schema and can be controlled by a programmer who does not know Teamcenter code or how to write it. A property set is simply a list of property set clauses.

Use PLM XML/TC XML Export Import Administration to create, edit, and delete property sets.

Property set clauses

Property set clauses have the following form:

{CLASS | TYPE}.*primary-class-or-type-name***:{PROPERTY | ATTRIBUTE } .***name-of-connecting-relationship***:{DO | SKIP}**

Clause item	Definition
Primary object selector	Specifies the primary object class or type to be translated. The class or type name is preceded by the **CLASS** or **TYPE** keyword and a period (**.**). For example: `CLASS.Item` For **PLM XML** export, the class or type hierarchy is not checked. Only the traversed objects with class or type matched exactly are honored.
Relation selector	Specifies the relationship between the secondary and primary object class or type. The relation name is preceded by a keyword and a period. For example: `ATTRIBUTE.items_tag` Using a wildcard (*****) rather than specifying a name after any of the keywords indicates that all relations of this type fit the rule. The following keywords apply to relation selectors:

Clause item	Definition	
	PROPERTY	Specifies the name of the Teamcenter property of the primary object that refers to the secondary object.
	ATTRIBUTE	Specifies the name of the Teamcenter attribute of the primary object that refers to the secondary object.
Action Type	Specifies the traversal actions to be taken. It must be one of the following keywords:	
	DO	Specifies that the action be executed.
	SKIP	Specifies that the relationship must not be followed. This directive is useful to eliminate special cases before a more general rule is reached. The **SKIP** action cannot be combined with any other action.

By default, all properties of a class in the property set are exported. You can exclude a property of the class by including a **SKIP** clause for the property. For example:

- Export all the properties of the **ItemRevision** class are exported except the **Archive Date** property:

  ```
  CLASS.ItemRevision:PROPERTY.archive_date:SKIP
  ```

- Export all properties of the **ItemRevision Master** form's storage class, if the form has data file, except the **User Data 1** property:

  ```
  TYPE.ItemRevision Master:PROPERTY.user_data_1:SKIP
  ```

If a Teamcenter object has a property set clause associated with it for the given transfer mode, a **UserData** element is created in the content of the equivalent PLM XML object the Teamcenter object gets translated into. The property or attribute appears as a PLM XML **UserValue** element in **UserData** with the value title set to the property name and the value set to the property or attribute value. If the property set looks like this:

```
CLASS.ItemRevision:PROPERTY.last_mod_date:DO
CLASS.ItemRevision:PROPERTY.object_string:DO
```

the PLM XML file looks like this:

```
<ProductRevision id="id12" name="assly" accessRefs="#id5" masterRef="#id20"
   revision="A">
     <UserData id="id15">
          <UserValue value="02-Dec-2003 12:37" title="last_mod_date"></UserValue>
          <UserValue value="000338/A-assly" title="object_string"></UserValue>
     </UserData>
```

```
    </ProductRevision>
```

This means that without writing any code, you have access to the Teamcenter property service. If the PLM XML object has a **UserData** element, the Teamcenter PLM XML import export framework tries to find a property with that name.

You can define property set clauses that leverage the Teamcenter class hierarchy to simplify the clauses in the Teamcenter PLM XML export. The behavior is as follows:

- For the properties defined on TYPE with a DO action, the property is exported.
- For the properties defined on a CLASS of the type or a parent CLASS of the type with a DO action, the property is exported.
- When the same property is defined on a TYPE or a CLASS or parent CLASS, the rules are as follows:
  - TYPE takes precedence over the CLASS or the parent CLASS of the type.
  - ASKIP takes precedence over a DO.

  This table provides the behavior for the various cases:

Class	Type	Parent class	Action	Description
DO	SKIP		SKIP	TYPE takes precedence over CLASS
SKIP	DO		DO	TYPE takes precedence over CLASS
DO		SKIP	SKIP	SKIP takes precedence over DO
SKIP		DO	SKIP	SKIP takes precedence over DO
	DO	SKIP	DO	TYPE takes precedence over parent CLASS
	SKIP	DO	SKIP	TYPE takes precedence over parent CLASS

Create property sets

1. Choose the **PropertySet** node in the **TransferMode** tree in the lower-left of the window.

   The system displays the **PropertySet** definition pane.

2. Type a name for the property set in the **Property Set Name** box.

3. Optionally, type a description of the property set in the **Description** box.

4. In **Scope of Property Set**, select either **Export** or **Import**.

5. Create the ordered clauses that specify how the data structure will be traversed:

    a. Click the **Add clause** button + located to the right of the clause table.

    b. Select one of the following keywords from the **Primary Object Class Type** list:

        CLASS

        TYPE (available only if **Scope of Property Set** is **Export**)

    c. Type the primary object's name in the **Primary Object** box.

    d. Select one of the following **Relation Type** options:

PROPERTY	Specifies the name of the Teamcenter property of the primary object that refers to the secondary object.
ATTRIBUTE	Specifies the name of the Teamcenter attribute of the primary object that refers to the secondary object.

    e. Type the name of a single property (or multiple properties) or objects in the **Related Property Or Object** box.

    f. Select one of the following from the **Property Action Type** list:

DO	Specifies that the action be executed.
SKIP	Specifies that the relationship must not be followed. This directive is useful to eliminate special cases before a more general rule is reached. The **SKIP** action cannot be combined with any other action.

    g. Repeat the previous steps to add additional clauses to the property set.

    h. Optionally, change the precedence of the set clauses by selecting a row in the table and using the **Move Up** ▲ or **Move Down** ▼ buttons to the right of the clause table.

6. Click the **Create** button.

    The system displays the new set in the **PropertySet** node of the **TransferMode** tree. This set can now be used when building transfer mode objects.

Edit property sets

You can edit property sets by adding, removing, or changing the definition of the clauses. You can also change the precedence of the clauses.

1. Select the node in the **PropertySet** branch that corresponds to the rule you want to modify.

2. Modify the contents or precedence of the clauses, as described in *Create property sets*.

3. Click the **Modify** button.

   The system saves the changes to the property set to the database.

Delete property sets

1. Select the node in the **PropertySet** branch that corresponds to the set you want to delete.

2. Click the **Delete** button.

   The system displays the **Delete Confirmation** dialog box.

3. Click the **Yes** button to delete the set, or click the **No** button to cancel.

Managing transfer mode objects

Transfer mode objects

Transfer mode objects combine rules and property sets to define the context of the PLM XML import or export operation. These objects are presented as context options when you import or export objects or system data using the Teamcenter applications.

Create transfer mode objects

1. Choose the **TransferMode** node in the **TransferMode** tree (located in the lower left pane of the PLM XML/TC XML Export Import Administration window).

   The system displays the **TransferMode** pane.

2. Type a unique name for the transfer mode object in the **Name** box. This name is displayed in the context fields of the Teamcenter export and import dialog boxes and is meant to clearly identify the context to the user.

3. Optionally, type the **context string** that maps the transfer mode object to a customized processor for the given object type in the **Context** box.

4. Type a description of the transfer mode object in the **Description** box.

5. Select a **Type of Transfer** option to specify whether this transfer mode applies to export or import.

6. From the **Output Schema Format** menu, choose either **PLMXML** for a standard PLM XML import or export or **TC XML** if you are using other Teamcenter data sharing methods to move data between Teamcenter and Teamcenter Enterprise sites.

7. Optionally, select the **Support Incremental** check box.

   This option allows updates to existing data during PLM XML import. For example, if an item being imported from an XML file that already exists in the database and **support incremental** is selected, the PLM XML import updates the item. If support incremental is not selected, the updates from the *.xml* file are ignored.

   This option is not available for the following classes. To update objects of these classes, use the **plmxml_import** utility with the **-import_mode=overwrite** argument.

Teamcenter class name	SDK class name
TransferMode	plmxml60::TransferMode
ClosureRule	plmxml60::ClosureRule
PropertySet	plmxml60:: PropertySet
Filter	plmxml60::FilterRule
PIEActionRule	Exported as **UserData** under **plmxml60::TransferMode**
Person	plmxml60::Person
TCCalendar	plmxml60::Calendar
User	plmxml60::User
Group	plmxml60::Organisation
Discipline	plmxml60::Discipline
Role	plmxml60::Role
POM_imc	plmxml60::Site
RevisionRule	plmxml60::RevisionRule
ListOfValues	plmxml60::ListOfValues
ImanQuery	plmxml60::SavedQueryDef

8. Select a rule from the **Closure Rule** list.

9. (Optional) Select a rule from the **Filter Rule** list.

10. (Optional) Select a property set from the **Property Set** list.

The details of a rule or property set can be viewed by selecting the node corresponding to the rule or property set in the **TransferMode** tree located in the lower-left portion of the window. The rule clauses are displayed in the right portion of the window.

11. (Optional) Select a revision rule from the **Revision Rule** list.

    A revision rule can be selected, but it is not applied to the target data during export and import.

12. (Optional) Select a tool from the **List of defined tools** and add it to the **List of selected tools**.

13. Click the **Create** button.

    The system saves the transfer mode object to the database and it is displayed in the **TransferMode** tree.

Edit transfer mode objects

You can edit transfer mode objects by adding, removing, or changing the rules, property sets, configuration objects, or actions that define the object.

1. Select the node in the **TransferMode** tree that corresponds to the object you want to edit.

2. Modify the definition by changing options, as described in *Create transfer mode objects*.

3. Click the **Modify** button.

   The system saves the changes to the object to the database.

Delete transfer mode objects

1. Select the node in the **TransferMode** tree that corresponds to the object you want to delete.

2. Click the **Delete** button.

   The system displays the **Delete Confirmation** dialog box.

3. Click the **Yes** button to delete the object, or click the **No** button to cancel.

Export transfer mode example

The following example incrementally builds the rules to export an assembly with its components. The assembly is an item named **assy1** that consists of two other items (**comp1** and **comp2**). The assembly item revision has an **MS PowerPoint** dataset called **sampleppt** with the **template.ppt** file attached. In this example, the items **assy1**, **comp1**, and **comp2** have been created in My Teamcenter and the assembly, in Structure Manager.

1. Create a closure rule with the name **sampleCR** and select **Export** for **Scope of Traversal**. Do not add any clauses yet. For step-by-step instructions, see *Create closure rules*.

2. Create a transfer mode with the name **sampleExport**. Type **DEFAULT_PIE_CONTEXT_STRING** in the **Context** box. Select **Export** for **Type of Transfer** and select your newly created **sampleCR** closure rule.

3. Using My Teamcenter, export the item revision for **assy1** using your new transfer mode. The result is similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="6" date="2006-10-06" time="15:58:36"
author="Teamcenter
  - smithj@IMC-433969693(433969693)">
<Header id="id1" traverseRootRefs="#id2" transferContext="sampleExport">
  </Header>
<ProductRevision id="id2" name="Assy1" accessRefs="#id3"
  subType="ItemRevision" masterRef="#id6" revision="A">
<ApplicationRef version="B_CVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef></ProductRevision>
<Product id="id6" name="Assy1" accessRefs="#id3" subType="Item"
  productId="000001">
<Description>a sample assembly</Description>
<ApplicationRef version="BmNVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
</Product>
<AccessIntent id="id3" intent="reference" ownerRefs="#id4"></
AccessIntent>
<Site id="id4" name="IMC-433969693" siteId="433969693">
<UserData id="id5">
<UserValue value="" title="connect_string"></UserValue>
<UserValue value="1" title="dbms"></UserValue>
<UserValue value="" title="imc_node_name"></UserValue>
<UserValue value="0" title="imc_ods_site"></UserValue>
</UserData>
</Site>
</PLMXML>
```

4. Add the following clause to the closure rule to export the item revision primary form:

   - **Primary Object Class Type**

     CLASS

   - **Primary Object**

     ItemRevision

- **Secondary Object Class Type**

  CLASS

- **Secondary Object**

  Form

- **Relation Type**

  RELATIONP2S

- **Related Property Or Object**

  IMAN_master_form

- **Action Type**

  PROCESS

5. Export the same item revision again. It looks similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="6" date="2006-10-06" time="15:58:36"
author="Teamcenter
  - smithj@IMC-433969693(433969693)">
<Header id="id1" traverseRootRefs="#id2" transferContext="sampleExport">
  </Header>
<ProductRevision id="id2" name="Assy1" accessRefs="#id3"
  subType="ItemRevision" masterRef="#id6" revision="A">
<ApplicationRef version="B_CVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
<AssociatedForm id="id9" role="IMAN_master_form" formRef="#id7">
</AssociatedForm>
</ProductRevision>
<Product id="id6" name="Assy1" accessRefs="#id3" subType="Item"
  productId="000001">
<Description>a sample assembly</Description>
<ApplicationRef version="BmNVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
</Product>
<AccessIntent id="id3" intent="reference" ownerRefs="#id4"></
AccessIntent>
<Site id="id4" name="IMC-433969693" siteId="433969693">
<UserData id="id5">
<UserValue value="" title="connect_string"></UserValue>
```

```
<UserValue value="1" title="dbms"></UserValue>
<UserValue value="" title="imc_node_name"></UserValue>
<UserValue value="0" title="imc_ods_site"></UserValue>
</UserData>
</Site>
<Form id="id7" name="000001/A" accessRefs="#id3"
  subType="ItemRevision Master" subClass="ItemRevision Master">
<ApplicationRef version="B_GVid3PR2drRD" application="Teamcenter"
  label="B_GVid3PR2drRD"></ApplicationRef>
<UserData id="id8" type="FormAttributes">
<UserValue value="" title="user_data_1"></UserValue>
<UserValue value="" title="user_data_2"></UserValue>
<UserValue value="" title="user_data_3"></UserValue>
<UserValue value="" title="previous_version_id"></UserValue>
<UserValue value="" title="serial_number"></UserValue>
<UserValue value="" title="project_id"></UserValue>
<UserValue value="" title="item_comment"></UserValue>
</UserData>
</Form>
</PLMXML>
```

The **AssociatedForm** element in the **ProductRevision** and the **Form** element is added to the previous export.

6. Add another clause to the closure rule to export specification attachments:

   - **Primary Object Class Type**

     CLASS

   - **Primary Object**

     ItemRevision

   - **Secondary Object Class Type**

     CLASS

   - **Secondary Object**

     Dataset

   - **Relation Type**

     RELATIONP2S

   - **Related Property Or Object**

IMAN_specification

- **Action Type**

  PROCESS

7. Export the same item revision again. It looks similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="6" date="2006-10-06" time="15:58:36"
author="Teamcenter
  - smithj@IMC-433969693(433969693)">
<Header id="id1" traverseRootRefs="#id2" transferContext="sampleExport">
  </Header>
<ProductRevision id="id2" name="Assy1" accessRefs="#id3"
  subType="ItemRevision" masterRef="#id6" revision="A">
<ApplicationRef version="B_CVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
<AssociatedForm id="id9" role="IMAN_master_form" formRef="#id7">
  </AssociatedForm>
<AssociatedDataSet id="id11" dataSetRef="#id10"
role="IMAN_specification">
  </AssociatedDataSet>
</ProductRevision>
<Product id="id6" name="Assy1" accessRefs="#id3" subType="Item"
  productId="000001">
<Description>a sample assembly</Description>
<ApplicationRef version="BmNVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
</Product>
<AccessIntent id="id3" intent="reference" ownerRefs="#id4">
</AccessIntent>
<Site id="id4" name="IMC-433969693" siteId="433969693">
<UserData id="id5">
<UserValue value="" title="connect_string"></UserValue>
<UserValue value="1" title="dbms"></UserValue>
<UserValue value="" title="imc_node_name"></UserValue>
<UserValue value="0" title="imc_ods_site"></UserValue>
</UserData>
</Site>
<Form id="id7" name="000001/A" accessRefs="#id3"
  subType="ItemRevision Master" subClass="ItemRevision Master">
<ApplicationRef version="B_GVid3PR2drRD" application="Teamcenter"
  label="B_GVid3PR2drRD"></ApplicationRef>
<UserData id="id8" type="FormAttributes">
<UserValue value="" title="user_data_1"></UserValue>
<UserValue value="" title="user_data_2"></UserValue>
```

```
<UserValue value="" title="user_data_3"></UserValue>
<UserValue value="" title="previous_version_id"></UserValue>
<UserValue value="" title="serial_number"></UserValue>
<UserValue value="" title="project_id"></UserValue>
<UserValue value="" title="item_comment"></UserValue>
</UserData>
</Form>
<DataSet id="id10" name="sampleppt" accessRefs="#id3" version="1"
  type="MSPowerPoint">
<ApplicationRef version="RSGVid3PR2drRD" application="Teamcenter"
  label="RSGVid3PR2drRD"></ApplicationRef>
</DataSet>
</PLMXML>
```

The **AssociatedDataset** element in the **ProductRevision** and the **Dataset** element is added to the previous export.

8. Add more clauses to the closure rule to export the associated files:

   Third clause

   - **Primary Object Class Type**

     TYPE

   - **Primary Object**

     MSPowerPoint

   - **Secondary Object Class Type**

     CLASS

   - **Secondary Object**

     ImanFile

   - **Relation Type**

     PROPERTY

   - **Related Property Or Object**

     ref_list

   - **Action Type**

PROCESS+TRAVERSE

Fourth clause

- **Primary Object Class Type**

  TYPE

- **Primary Object**

  MSExcel

- **Secondary Object Class Type**

  CLASS

- **Secondary Object**

  ImanFile

- **Relation Type**

  PROPERTY

- **Related Property Or Object**

  ref_list

- **Action Type**

  PROCESS+TRAVERSE

Fifth clause

- **Primary Object Class Type**

  CLASS

- **Primary Object**

  ImanFile

- **Secondary Object Class Type**

  CLASS

- **Secondary Object**

  *

- **Relation Type**

  ATTRIBUTE

- **Related Property Or Object**

  original_file_name

- **Action Type**

  PROCESS

In addition, change the **Action Type** for the second clause (the one exporting the dataset) to **PROCESS+TRAVERSE**.

9. Export the same item revision again. It looks similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="6" date="2006-10-06" time="15:58:36"
author="Teamcenter
  - smithj@IMC-433969693(433969693)">
<Header id="id1" traverseRootRefs="#id2" transferContext="sampleExport">
  </Header>
<ProductRevision id="id2" name="Assy1" accessRefs="#id3"
  subType="ItemRevision" masterRef="#id6" revision="A">
<ApplicationRef version="B_CVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
<AssociatedForm id="id9" role="IMAN_master_form" formRef="#id7">
</AssociatedForm>
<AssociatedDataSet id="id11" dataSetRef="#id10"
role="IMAN_specification">
</AssociatedDataSet>
</ProductRevision>
<Product id="id6" name="Assy1" accessRefs="#id3" subType="Item"
  productId="000001">
<Description>a sample assembly</Description>
<ApplicationRef version="BmNVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
</Product>
<AccessIntent id="id3" intent="reference" ownerRefs="#id4"></
AccessIntent>
<Site id="id4" name="IMC-433969693" siteId="433969693">
<UserData id="id5">
```

```
<UserValue value="" title="connect_string"></UserValue>
<UserValue value="1" title="dbms"></UserValue>
<UserValue value="" title="imc_node_name"></UserValue>
<UserValue value="0" title="imc_ods_site"></UserValue>
</UserData>
</Site>
<Form id="id7" name="000001/A" accessRefs="#id3"
  subType="ItemRevision Master" subClass="ItemRevision Master">
<ApplicationRef version="B_GVid3PR2drRD" application="Teamcenter"
  label="B_GVid3PR2drRD"></ApplicationRef>
<UserData id="id8" type="FormAttributes">
<UserValue value="" title="user_data_1"></UserValue>
<UserValue value="" title="user_data_2"></UserValue>
<UserValue value="" title="user_data_3"></UserValue>
<UserValue value="" title="previous_version_id"></UserValue>
<UserValue value="" title="serial_number"></UserValue>
<UserValue value="" title="project_id"></UserValue>
<UserValue value="" title="item_comment"></UserValue>
</UserData>
</Form>
<DataSet id="id10" name="sampleppt" accessRefs="#id3" version="1"
  type="MSPowerPoint">
<ApplicationRef version="RSGVid3PR2drRD" application="Teamcenter"
  label="RSGVid3PR2drRD"></ApplicationRef>
</DataSet>
<ExternalFile id="id12" accessRefs="#id3"
  locationRef="000001_A-Assy1\template.ppt" format="ppt">
<ApplicationRef version="BfKVid3PR2drRD" application="Teamcenter"
  label="BfKVid3PR2drRD"></ApplicationRef>
</ExternalFile>
</PLMXML>
```

The **ExternalFile** element is for **MSPowerPoint** only.

10. Add more clauses to the closure rule to export the structure from Structure Manager:

    Sixth clause

    - **Primary Object Class Type**

      TYPE

    - **Primary Object**

      BOMLine

    - **Secondary Object Class Type**

TYPE

- **Secondary Object**

  BOMLine

- **Relation Type**

  PROPERTY

- **Related Property Or Object**

  bl_child_lines

- **Action Type**

  PROCESS+TRAVERSE

Seventh clause

- **Primary Object Class Type**

  TYPE

- **Primary Object**

  BOMLine

- **Secondary Object Class Type**

  CLASS

- **Secondary Object**

  ItemRevision

- **Relation Type**

  PROPERTY

- **Related Property Or Object**

  bl_revision

- **Action Type**

PROCESS+TRAVERSE

11. Export the assembly related to the same item revision from Structure Manager. It looks similar to this:

```
<?xml version="1.0" encoding="utf-8"?>
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="6" date="2006-10-06" time="15:58:36"
author="Teamcenter
  - smithj@IMC-433969693(433969693)">
<Header id="id1" traverseRootRefs="#id6" transferContext="sampleExport">
</Header>
<ProductRevision id="id11" name="comp1" accessRefs="#id12"
  subType="ItemRevision" masterRef="#id15" revision="A">
<ApplicationRef version="RKDVid3PR2drRD" application="Teamcenter"
  label="RGKVid3PR2drRD"></ApplicationRef>
<AssociatedForm id="id18" role="IMAN_master_form" formRef="#id16">
</AssociatedForm>
<AssociatedDataSet id="id20" dataSetRef="#id19"
role="IMAN_specification">
</AssociatedDataSet></ProductRevision>
<ProductRevision id="id24" name="comp2" accessRefs="#id12"
  subType="ItemRevision" masterRef="#id25" revision="A">
<ApplicationRef version="ROKVid3PR2drRD" application="Teamcenter"
  label="ROBVid3PR2drRD"></ApplicationRef>
<AssociatedForm id="id28" role="IMAN_master_form" formRef="#id26">
</AssociatedForm></ProductRevision>
<ProductRevision id="id29" name="Assy1" accessRefs="#id12"
  subType="ItemRevision" masterRef="#id30" revision="A">
<ApplicationRef version="B_CVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef>
<AssociatedForm id="id33" role="IMAN_master_form" formRef="#id31">
</AssociatedForm>
<AssociatedDataSet id="id35" dataSetRef="#id34"
role="IMAN_specification">
</AssociatedDataSet></ProductRevision>
<Product id="id15" name="comp1" accessRefs="#id12" subType="Item"
  productId="000002">
<ApplicationRef version="RGKVid3PR2drRD" application="Teamcenter"
  label="RGKVid3PR2drRD"></ApplicationRef></Product>
<Product id="id25" name="comp2" accessRefs="#id12" subType="Item"
  productId="000003">
<ApplicationRef version="ROBVid3PR2drRD" application="Teamcenter"
  label="ROBVid3PR2drRD"></ApplicationRef></Product>
<Product id="id30" name="Assy1" accessRefs="#id12" subType="Item"
  productId="000001">
<Description>a sample assembly</Description>
<ApplicationRef version="BmNVid3PR2drRD" application="Teamcenter"
  label="BmNVid3PR2drRD"></ApplicationRef></Product>
```

```
<RevisionRule id="id2" name="Latest Working">
<Description>Latest Working else Latest Any Status</Description>
<ApplicationRef version="o3IVRSyxR2drRD" application="Teamcenter"
  label="o3IVRSyxR2drRD"></ApplicationRef></RevisionRule>
<ProductView id="id4" ruleRefs="#id2" rootRefs="id6"
  primaryOccurrenceRef="id6">
<ApplicationRef application="Teamcenter"
  label="hWKVid3PR2drRD/o3IVRSyxR2drRD/AAAAAAAAAAAAAA/BOM">
</ApplicationRef>
<UserData id="id3" type="TC Specific Properties">
<UserValue value="imprecise" title="BOM_precision_type"></UserValue>
</UserData>
<Occurrence id="id6" instancedRef="#id29" occurrenceRefs="id9 id22">
<ApplicationRef application="Teamcenter" label="hWKVid3PR2drRD/">
</ApplicationRef>
<UserData id="id7" type="AttributesInContext">
<UserValue value="" title="AO_ID"></UserValue>
<UserValue value="" title="SequenceNumber"></UserValue>
<UserValue value="" title="OccurrenceName"></UserValue>
<UserValue value="" title="Quantity"></UserValue></UserData>
<Transform id="id5">1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</Transform>
</Occurrence>
<Occurrence id="id9" instancedRef="#id11" parentRef="#id6">
<ApplicationRef application="Teamcenter"
  label="hWKVid3PR2drRD/heMVid3PR2drRD/"></ApplicationRef>
<UserData id="id10" type="AttributesInContext">
<UserValue value="" title="AO_ID"></UserValue>
<UserValue value="10" title="SequenceNumber"></UserValue>
<UserValue value="" title="OccurrenceName"></UserValue>
<UserValue value="" title="Quantity"></UserValue></UserData>
<Transform id="id8">1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</Transform>
</Occurrence>
<Occurrence id="id22" instancedRef="#id24" parentRef="#id6">
<ApplicationRef application="Teamcenter"
  label="hWKVid3PR2drRD/hiEVid3PR2drRD/"></ApplicationRef>
<UserData id="id23" type="AttributesInContext">
<UserValue value="" title="AO_ID"></UserValue>
<UserValue value="20" title="SequenceNumber"></UserValue>
<UserValue value="" title="OccurrenceName"></UserValue>
<UserValue value="" title="Quantity"></UserValue></UserData>
<Transform id="id21">1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</Transform>
</Occurrence></ProductView>
<AccessIntent id="id12" intent="reference" ownerRefs="#id13"></
AccessIntent>
<Site id="id13" name="IMC-433969693" siteId="433969693">
<UserData id="id14">
<UserValue value="" title="connect_string"><</UserValue>
<UserValue value="1" title="dbms"><</UserValue>
<UserValue value="" title="imc_node_name"></UserValue>
```

```
<UserValue value="0" title="imc_ods_site"></UserValue></UserData></Site>
<Form id="id16" name="000002/A" accessRefs="#id12"
  subType="ItemRevision Master" subClass="ItemRevision Master">
<ApplicationRef version="RKGVid3PR2drRD" application="Teamcenter"
  label="RKGVid3PR2drRD"></ApplicationRef>
<UserData id="id17" type="FormAttributes">
<UserValue value="" title="user_data_1"></UserValue>
<UserValue value="" title="user_data_2"></UserValue>
<UserValue value="" title="user_data_3"></UserValue>
<UserValue value="" title="previous_version_id"></UserValue>
<UserValue value="" title="serial_number"></UserValue>
<UserValue value="" title="project_id"></UserValue>
<UserValue value="" title="item_comment"></UserValue></UserData></Form>
<DataSet id="id19" name="sampleexcel" accessRefs="#id12" version="1"
  type="MSExcel">
<ApplicationRef version="RaGVid3PR2drRD" application="Teamcenter"
  label="RaGVid3PR2drRD"></ApplicationRef></DataSet>
<Form id="id26" name="000003/A" accessRefs="#id12"
  subType="ItemRevision Master" subClass="ItemRevision Master">
<ApplicationRef version="RONVid3PR2drRD" application="Teamcenter"
  label="RONVid3PR2drRD"></ApplicationRef>
<UserData id="id27" type="FormAttributes">
<UserValue value="" title="user_data_1"></UserValue>
<UserValue value="" title="user_data_2"></UserValue>
<UserValue value="" title="user_data_3"></UserValue>
<UserValue value="" title="previous_version_id"></UserValue>
<UserValue value="" title="serial_number"></UserValue>
<UserValue value="" title="project_id"></UserValue>
<UserValue value="" title="item_comment"></UserValue></UserData></Form>
<Form id="id31" name="000001/A" accessRefs="#id12"
  subType="ItemRevision Master" subClass="ItemRevision Master">
<ApplicationRef version="B_GVid3PR2drRD" application="Teamcenter"
  label="B_GVid3PR2drRD"></ApplicationRef>
<UserData id="id32" type="FormAttributes">
<UserValue value="" title="user_data_1"></UserValue>
<UserValue value="" title="user_data_2"></UserValue>
<UserValue value="" title="user_data_3"></UserValue>
<UserValue value="" title="previous_version_id"></UserValue>
<UserValue value="" title="serial_number"></UserValue>
<UserValue value="" title="project_id"></UserValue>
<UserValue value="" title="item_comment"></UserValue></UserData></Form>
<DataSet id="id34" name="sampleppt" accessRefs="#id12" version="2"
  memberRefs="#id36" type="MSPowerPoint">
<ApplicationRef version="BbMVid3PR2drRD" application="Teamcenter"
  label="BbMVid3PR2drRD">
</ApplicationRef></DataSet>
<ExternalFile id="id36" accessRefs="#id12"
  locationRef="000001_A-Assy1\template.ppt" format="ppt">
<ApplicationRef version="BfKVid3PR2drRD" application="Teamcenter"
```

```
  label="BfKVid3PR2drRD">
</ApplicationRef></ExternalFile>
</PLMXML>
```

The structure information is exported to the file. The related item revisions and their attachments are also exported.

Tip:

If you are not sure about the details of the data model and need to find the right reference, refer to the object properties in My Teamcenter or Structure Manager to find the property for the related object.

Transfer modes included with Teamcenter

The following transfers modes are included with Teamcenter. Additional transfer modes may be available depending on the optional applications that you install. For information about application-specific transfer modes, refer to the documentation for your application.

A transfer mode can optionally have a context string to determine the export/import behavior of traversed objects.

Transfer mode [Context string]	Import or export	Description
assignment_list_export	Export	Exports process assignment lists.
BOMwriterExport [**BOMWRITER_PIE_CONTEXT_STRING**]	Export	Exports bill of materials lists. Do not use this transfer mode with the **plmxml_import** and **plmxml_export** utilities or in the client interface. This transfer mode is used by Teamcenter internally.
ConfiguredDataExportDefault [**DEFAULT_PIE_CONTEXT_STRING**]	Export	Exports configured BOM assemblies, including **ImanItem** lines, forms, datasets, folders, occurrences, activities, and appearance paths. BOP processes, GDE, activities, variant expressions, and naming rules are also supported. This transfer mode works the same as the **ConfiguredDataFiles ExportDefault** transfer mode but does not export explicit file contents.
ConfiguredDataFilesExportDefault [**DEFAULT_PIE_CONTEXT_STRING**]	Export	Exports configured BOM assemblies, including **ImanItem** lines, forms, datasets, actual files, folders, occurrences, activities, and appearance paths. There is equal support for BOP processes,

Transfer mode [Context string]	**Import or export**	**Description**
		GDE, activities, variant expressions, and naming rules.
ConfiguredDataImportDefault [DEFAULT_PIE_CONTEXT_STRING]	Import	Imports configured BOM assemblies, including structure, forms, datasets, folders, occurrences, predecessors, activities, folders, variant conditions, and transformations.
ConfiguredRequirementDataExport [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports trace links on occurrences.
ctm0_ContMgmtExportAdmin [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports Content Management objects.
ctm0_ContMgmtImportAdmin [ctm0_ContMgmtImportAdmin]	Import	Imports Content Management objects.
ExportGraphic	Export	Exports graphic files.
ExportProductView [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports BOM window and BOM line-centric assemblies, configured attachments, data sets, appearance nodes, and manufacturing data. There are several clauses that can be activated for tuning exports and controlling output.
ExportValidationResult [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports validation objects.
incremental import and **incremental_import** [DEFAULT_PIE_CONTEXT_STRING]	Import	**ProductView** is imported and the **InstanceGraph** is skipped. All other elements and attributes are also imported. The **incremental_import** transfer mode (with an underscore in place of a space) is used on Linux platforms. Either transfer mode can be used on Windows.
ICSExportSubtreeWithWSOs [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports the entire hierarchy with content.
MechatronicsFoundationDataExport [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports collaborative context for CAD-centric data, including BOM, assembly, configured attachments, bill of process, operations, **TypeCannedMethod**, naming rules, GDE links, variant expressions, signals, EPM tasks, and EPM jobs.
MRMAssemblyExport [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports Resource Manager assemblies.
PLMXMLAdminDataExport [DEFAULT_TIE_CONTEXT]	Export	Exports transfer modes and transfer option sets in TC XML format.

Transfer mode [Context string]	Import or export	Description
		Do not use this transfer mode with the **plmxml_import** and **plmxml_export** utilities.
ScheduleManagerFndImportLegacy [DEFAULT_PIE_CONTEXT_STRING]	Import	Imports an out-of-the-box schedule and its related data, including properties.
ScheduleManagerFoundationExport [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports an out-of-the-box schedule and its related data, including properties.
ScheduleManagerFoundationImport [DEFAULT_PIE_CONTEXT_STRING]	Import	Imports an out-of-the-box schedule and its related data, including properties.
TIEEdaLibExport [DEFAULT_TIE_CONTEXT]	Export	Exports the Electronic Design Automation (EDA) library. Do not use this transfer mode with the **plmxml_import** and **plmxml_export** utilities.
TIEImportDefault [DEFAULT_TIE_CONTEXT]	Import	Imports TC XML data. This is the default transfer mode for importing TC XML data.
TIEUnconfiguredExportDefault	Export	Exports Teamcenter data. This is the default transfer mode for exporting Teamcenter data. Do not use this transfer mode with the **plmxml_import** and **plmxml_export** utilities.
TR_AUDIT_REPORT [AUDIT_REPORT_CONTEXT]	Export	Exports EPM jobs, signoffs, tasks, items, item revisions, and file contents.
TL_Import [DEFAULT_REQ_CONTEXT_STRING]	Import	Imports trace links on occurrences.
TransferGenericObject_TM [PWT_PIE_CONTEXT_STRING]	Export	Exports architecture, including many relationships from secondary to primary, including **EC_affected_item_rel**, **EC_solution_item_rel**, process staging, and variant expressions. There are several specific clause rules for architecture design and part usage.
transgde [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports attachments, product structure, appearance groups, bill of process, activities, and variant expressions. It is design element-centric and supports **CCObject** and structure context.
unconfiguredDataFileExport [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports unconfigured items, item revisions, datasets, and **ImanFile** contents.
web_reports [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports reports.
workflow_template_import [DEFAULT_PIE_CONTEXT_STRING]	Import	Imports workflow templates and process assignment lists. This transfer mode contains

Transfer mode [Context string]	Import or export	Description
		open-ended closure rules (*) for any class, any attribute, and all contents.
workflow_template_mode [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports workflow task templates, signoffs, dependencies on tasks, business rule handlers, action handlers, and actions.
workflow_template_overwrite [DEFAULT_PIE_CONTEXT_STRING]	Import	Overwrites existing workflow templates for mass import. This transfer mode contains open-ended closure rules (*) for any class, any attribute, and all contents.
write_to_technomatix [DEFAULT_PIE_CONTEXT_STRING]	Export	Exports plant, process, and product data to Tecnomatix. This transfer mode is manufacturing engineering-centric and is used to export attachments, product structure, appearance path nodes, bill of process, and operations per work area.

Update transfer modes

Changes to the infrastructure require you to import the transfer modes and modify the closure rules.

1. Import the transfer modes as follows:

   - For transfer mode **.xml** files in PLM XML format, use the **plmxml_import** utility with the **-import_mode=overwrite** argument.

   - For transfer mode **.xml** files in TC XML format, use the **tcxml_import** utility with the **-scope_rules_mode=overwrite** argument.

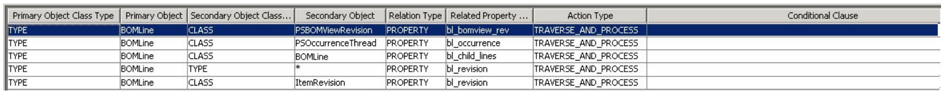
2. For all custom PLM XML transfer modes and any customized out-of-the-box transfer modes, modify the closure rules as follows:

   a. Remove any closure rule with **TYPE.ImanItemLine** or **CLASS.ImanItemLine** as the primary or secondary object selector. These unused closure rule clauses from previous versions of Teamcenter must be removed before they can be redefined.

   b. Change any closure rule with **TYPE.BOMLine** as the primary or secondary (or both) object selector to **CLASS.ImanItemLine**. Then, add the **PRIMARY.bl_window_is_BOPWin==”false”** or **SECONDARY.bl_window_is_BOPWin==”false”** conditional clause to the appropriate closure rule (primary or secondary (or both) object selector, respectively).

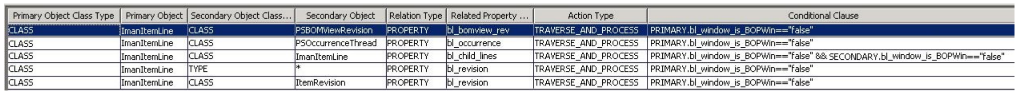
      This indicates that the rule applies to every subtype of **ImanItemLine** if the line is not found in **BOPWindow**.

The following is an example of a closure rule that needs updating.



Primary Object Class Type	Primary Object	Secondary Object Class...	Secondary Object	Relation Type	Related Property ...	Action Type	Conditional Clause
TYPE	BOMLine	CLASS	PSBOMViewRevision	PROPERTY	bl_bomview_rev	TRAVERSE_AND_PROCESS	
TYPE	BOMLine	CLASS	PSOccurrenceThread	PROPERTY	bl_occurrence	TRAVERSE_AND_PROCESS	
TYPE	BOMLine	CLASS	BOMLine	PROPERTY	bl_child_lines	TRAVERSE_AND_PROCESS	
TYPE	BOMLine	TYPE	*	PROPERTY	bl_revision	TRAVERSE_AND_PROCESS	
TYPE	BOMLine	CLASS	ItemRevision	PROPERTY	bl_revision	TRAVERSE_AND_PROCESS	

Following is what it should look like after it is updated.



Primary Object Class Type	Primary Object	Secondary Object Class...	Secondary Object	Relation Type	Related Property ...	Action Type	Conditional Clause
CLASS	ImanItemLine	CLASS	PSBOMViewRevision	PROPERTY	bl_bomview_rev	TRAVERSE_AND_PROCESS	PRIMARY.bl_window_is_BOPWin=="false"
CLASS	ImanItemLine	CLASS	PSOccurrenceThread	PROPERTY	bl_occurrence	TRAVERSE_AND_PROCESS	PRIMARY.bl_window_is_BOPWin=="false"
CLASS	ImanItemLine	CLASS	ImanItemLine	PROPERTY	bl_child_lines	TRAVERSE_AND_PROCESS	PRIMARY.bl_window_is_BOPWin=="false" && SECONDARY.bl_window_is_BOPWin=="false"
CLASS	ImanItemLine	TYPE	*	PROPERTY	bl_revision	TRAVERSE_AND_PROCESS	PRIMARY.bl_window_is_BOPWin=="false"
CLASS	ImanItemLine	CLASS	ItemRevision	PROPERTY	bl_revision	TRAVERSE_AND_PROCESS	PRIMARY.bl_window_is_BOPWin=="false"

c. Change any closure rule with **TYPE.ImanItemBOPLine** as the primary or secondary (or both) object selector to **CLASS.ImanItemBOPLine**. Then, add the **PRIMARY.bl_me_appgroup==”0”** or **SECONDARY.bl_me_appgroup==”0”** conditional clause to the appropriate closure rule (primary and/or secondary object selector, respectively).

   This indicates that the rule applies to every subtype of **ImanItemBOPLine** if the line does not reflect an appearance group.

d. Change any closure rule with **TYPE.AppGrpBOPLine** as the primary or secondary object selector to **CLASS.AppGrpBOPLine**.

   This indicates that the rule applies to every subtype of **AppGrpBOPLine**.

e. In all closure rules with constraint checking, change the **ImanItemP** property to **Item** and the **ImanItemRevP** property to **ItemRevision**.

Export and Import rules included with Teamcenter

The following filter rules are included with the PLM XML/TC XML Export Import Administration application.

Filter rule name	Exchange type	Description
CAEImportFilterRule	PLM XML import	Skips the **BOMViewRevision** that are not of CAE Analysis type.
CLS_PIE_NodeHierarchyDefault	PLM XML export	Skips sibling hierarchy nodes of a candidate **Hierarchy Node** while traversing from its parent, using children traversal property.
ExportDatasetFilter	PLM XML export	Determines if a CAD file will be exported when the CAD file is owned by an **ItemRevision** or an **Attachment Window**. A list of preferred dataset types is defined by the preference **PreferPlmxmlCadFiles** with precedence from high to low.

Filter rule name	Exchange type	Description
ExportLatestObjectFilter	PLM XML export	Refreshes the object to get the latest data to export.
MySampleFilter	PLM XML	A sample filter to always process the object.
CLS_TIE_NodeHierarchyDefault	TC XML export	If the root or selected object is a **Library** or a **Library Hierarchy**, sibling traversal of the **Hierarchy Node** is allowed. It also skips the **Ptn0ChildParentLink** if its child partition is not in the ancestral lineage of the selected **Hierarchy Node**.
ExportReleasedObjectFilter	TC XML export	Exports only released objects.
ExportReleasedPrimaryFilter	TC XML export	Exports only relations whose primary object is released.
MyTieSampleFilter	TC XML	A sample filter to always process the object.
TieBomLevelFilter	TC XML export	Exports a BOM until the BOM line level reaches the value defined by the **–bomlevel** option in **tcxml_export** utility.

The following action handlers are used with the PLM XML/TC XML Export Import Administration application action rules.

Action handler	Action location	Exchange type	Description
CAEExportPreAction	Pre Action	PLM XML export	Supported in a future release.
CAEExportDuringAction	During Action	PLM XML export	Inserts CAE global preference data and application preference data into the PLM XML document. It also checks out the CAE analysis type of **BOMViewRevision**.
CAEImportPostAction	Post Action	PLM XML import	Checks in the **BOMViewRevision** CAE analysis type.
IncrementalUpdatePreAction	Pre Action	PLM XML export	Used by Application Interface to export BOM lines that have changed since the previous export.
EnableItemRevReviseDeepCopy	During Action	PLM XML import	Enables **ItemRevision** deep copy when revising an **ItemRevision** during import.
SampleTIEPreAction	Pre Action	TC XML	A sample pre-action to always process the object.
SampleTIEDuringAction	During Action	TC XML	A sample during-action to always process the object.

Action handler	Action location	Exchange type	Description
SampleTIEPostAction	Post Action	TC XML	A sample post-action to always process the object.
SampleTMSPostAction	Post Action	TC XML import	A sample post-action to export multi-site import data information.
WorkflowImportPostActionRule	Post Action	TC XML import	For administrative data import, this handler finds the workflow template at the target site with the name or origin ID matching the ones in the imported template. It marks the imported or target site templates as obsolete based on the import merge option.

Using context strings

A context string is used with a transfer mode to determine the export/import behavior of traversed objects. Each object type has registered export/import functions with various context strings. One type object can have multiple export/import functions with different context strings and each function may have different export/import processing. During the export/import, only the export/import function that matches the context string of the applied transfer mode is executed.

When defining a transfer mode, a **Context** value is not required.

A custom context string can be created and used with a transfer mode. The custom context string definition includes registered export/import functions that are based on the object type. When the transfer mode with the custom context string is used, the export/import functions matching the object type are executed.

The following context strings are included with Teamcenter. Additional context strings may be available depending on the optional applications that you install. For information about application-specific context strings, refer to the documentation for your application.

Context string name	PLM XML / TC XML	Usage
DEFAULT_PIE_CONTEXT_STRING	PLM XML	Default PLM XML export/import
DEFAULT_REQ_CONTEXT_STRING	PLM XML	PLM XML import of trace links
DEFAULT_TIE_CONTEXT	TC XML	Default TC XML export/import
PROCESS_SIMULATE_CONTEXT_STRING	PLM XML	Export/import with Process Simulate
REQ_WORD_TEMPLATE_CONTEXT_STRING	PLM XML	Requirements Manager word template export/import
SERVICE_PLANNING_CONTEXT_STRING	PLM XML	Service Planning data export/import

Context string name	PLM XML / TC XML	Usage
TECNOMATIX_CONTEXT_STRING	PLM XML	Tecnomatix data export/import
VALIDATION_PIE_CONTEXT_STRING	PLM XML	Validation Manager data export/ import

Transfer mode best practices

Improving performance

Transfer modes

Transfer modes focus on specific data traversals. Modify transfer modes to tune them for better performance.

Export performance

You can improve the performance of your PLM XML exports by setting the **PLMXML_export_packed_bom_<transfer-mode-name>** preference to **TRUE**. Setting this preference to **TRUE** improves the PLM XML export performance for the specified transfer mode.

Caution:

If you export a packed BOM, you may lose data in the exported XML file. Therefore, do not import an XML file with a packed BOM into Teamcenter. Set this preference to **TRUE** only if you plan to use the XML file in a third-party application.

Transfer mode limitations

Do not use the **JTDataExportDefault** and **JTDataImportDefault** transfer modes to directly export or import JT files with the **plmxml_export** and **plmxml_import** utilities or the **PLM XML** option in the user interface. Instead, use the PLM XML format rather than the JT assembly format. This exports and imports JT files, along with the PLM XML file containing the item information and assembly relationships.

Further, do not use the following transfer modes with the **plmxml_export** and **plmxml_import** utilities:

- **BOMwriterExport**
- **PLMXMLAdminDataExport**
- **TIEImportDefault**
- **TIEPDXExportDefault**

- **TIEUnconfiguredExportDefault**

Use specific type identifiers for better performance

The following example describes a typical optimization of the closure rules in a transfer mode. This example is not complete and does not constitute a specific modification. The following closure rules are used in the **ConfiguredDataExportDefault** transfer mode:

```
TYPE.BOMLine : TYPE.* :PROPERTY.bl_attachments:TravAndProc:
     SECONDARY.al_source_class==Dataset
TYPE.BOMLine : TYPE.* :PROPERTY.bl_attachments:TravAndProc:
     SECONDARY.al_source_class==Form
TYPE.BOMLine : TYPE.* :PROPERTY.bl_attachments:TravAndProc:
     SECONDARY.al_source_class==Folder
```

These closure rules specify the following:

- Traverse and process any **SECONDARY** type from **BOMLine** to any type by using the **bl_attachments** property.
- Execute the **al_source_class** function on the **SECONDARY** type.
- Support its export if the function returns a dataset, form, or folder.

To optimize the closure rules, change the types to take advantage of specific type identifiers. The engine looks only at these types and not all types.

```
TYPE.BOMLine : TYPE.Dataset:PROPERTY.bl_attachments:TravAndProc
TYPE.BOMLine : TYPE.Form :PROPERTY.bl_attachments:TravAndProc
TYPE.BOMLine : TYPE.Folder :PROPERTY.bl_attachments:TravAndProc
```

Unload objects in transfer mode to speed processing

You can use the **PIE_***transfer-mode-name***_unload_objects** and **PIE_***transfer-mode-name***_process_in_chunks** user preferences to improve performance by unloading objects no longer needed in memory. The **PIE_***transfer-mode-name***_unload_objects** preference tells Teamcenter which objects to unload from memory and in what order.

Caution:

If you unload objects required by other rich client functions or features (such as BOM lines), you may cause errors. Ensure that all objects are saved to the database before unloading.

When using **PIE_***transfer-mode-name***_unload_objects**, create a preference for each transfer mode type from which you want to unload objects and set the preference with values specifying object types that are to be unloaded. For example:

```
PIE_sampleExport_unload_objects=
ImanItemBOPLine
BOMLine
Item
ItemRevision
```

Caution:

These preferences define the processing order of the specified objects and the forced unloading of the same objects. (The first object listed as a value is processed first, the second is processed next, and so on.) If objects are unloaded in an improper order, the existence of dependent objects cannot be translated and the child objects are bypassed, even if closure rules specify the traversal of these objects. To prevent errors. be aware of which objects can be processed and unloaded first, and list the objects in the correct order.

If you set the **PIE_***transfer-mode-name***_unload_objects** preference, you can also use the **PIE_***transfer-mode-name***_process_in_chunks** preference to specify the objects that are to be unloaded in chunks or batches. The default is that all objects are unloaded in chunks.

Maintain COTS scope rules

COTS scope rules (transfer modes, closure rules, filter rules, and so on) are maintained in XML files in the *TC_DATA* directory (for example, **defaultTransfermodes.xml**). The file is imported to Teamcenter using the **tcxml_import** command line utility during the database installation and upgrade.

The transfer mode **.xml** files are imported in overwrite mode as part of the upgrade. You need not manually import the files. To reload the latest changes from the **.xml** file, you can run the **tcxml_import** utility in overwrite mode to ensure that the updates to existing scope rules are also imported to the database. For example:

```
$TC_ROOT/bin/tcxml_import -u=user -p=password -g=group -file=$TC_DATA/
defaultTransfermodes.xml -scope_rules -scope_rules_mode=overwrite
```

Siemens Digital Industries Software recommends that you do not modify the COTS scope rules. If necessary, you can create a copy of the COTS rules and modify the copy as needed.

Caution:

All references to obsolete attributes and classes must be removed from custom closure rules or property sets. Additionally, remove any deprecated attributes and classes from custom closure rules or property sets.

For information about deprecated and obsolete classes and attributes, see the latest Teamcenter **README** file in the **Downloads** area on Support Center.

Closure rule best practices

Use session rules to identify the first set of target objects

Closure rules that have **TYPE.PIESession** as the primary object selector are session rules. Session rules are evaluated only once during the first pass of closure rule clauses during the first step of the traversal process to identify the initial set of target objects. Afterword, the target objects are processed using the regular traversal mechanism and are evaluated against all clauses. The secondary object selector is evaluated for an exact match and includes all subclasses for **CLASS** or subtypes for **TYPE**.

For example, if the closure rule is **TYPE.PIESession:CLASS.Item : PROPERTY.target_list : SKIP:** and **Item** is the class name of the target object selected for export, the object matches the closure rule. Because the action type is **SKIP**, all **Item** objects are skipped, including all objects that are subclasses of **Item**, such as **Document**.

Export other ApplicationRef elements

The **ApplicationRef** element is a PLM XML construct that allows an application to identify an object by an application-specific identifier. Teamcenter always exports an **ApplicationRef** element for each Teamcenter object it exports. But the Teamcenter PLM XML translator can store the **ApplicationRef** elements of data coming from other applications. By default, these **ApplicationRef** elements are not written into the export file, but if you want to export them, use this clause:

```
CLASS.*:CLASS.PLMAppUID:PROPERTY.application_uid:PROCESS+TRAVERSE:
```

Though the **PLMAppUID** object does not have an **application_uid** property, it allows control of this type of translation.

Ignore ApplicationRef elements on import

Use the following clause to ignore ApplicationRef elements on import:

```
TYPE.PIESession:CLASS.ApplicationRef:CONTENT.*:SKIP
```

Avoid automatic check-in on import

Use the following clause to avoid automatic check-in on import:

```
TYPE.PIESession:CLASS.NO_AUTO_CHECK_IN:CONTENT.*:SKIP
```

By default, imported objects are automatically checked in.

Export substitutes

If you export a structure in which substitute components are defined, the substitutes are also exported if you add the following closure rule to your transfer mode:

```
TYPE.BOMLine:TYPE.BOMLine:PROPERTY.bl_alternate:TRAVERSE+PROCESS
```

Compare source and destination BOMs

Use the **AccountabilityMOA** closure rule to compare source and destination BOMs, traversing or skipping occurrences in the source structure based on the occurrence type of the linked occurrences in the destination structure. By default, the closure rule provides the **MEConsumed** occurrence type for processing linked occurrences in the destination structure and the **MEOther** occurrence type for skipping occurrences assigned from the EBOM.

Modify the default rule as necessary for your site. As an example, process **MEConsumed** and **MEAssigned** occurrence types as follows:

```
CLASS BOMLine CLASS BOMLine FUNCTION ebom_assy_assigned_only PROCESS
FUNC_ARGS("$target.bl_occ_type";"=";"MEConsumed";"||";"$source.bl_occ_typ
e";
"=";"MEConsumed";"||";"$source.bl_occ_type";"=";"MEAssign";"||";
"$target.bl_occ_type";"=";"MEAssign")&&
SECONDARY.bl_has_children=="true"&&SECONDARY.bl_load_state=="0"
```

To skip **MEOther** and **MEHandle** occurrence types, modify the clause as follows:

```
CLASS BOMLine CLASS BOMLine FUNCTION ebom_assy_assigned_only SKIP
FUNC_ARGS("$source.bl_occ_type";"=";"MEOther";"||";
"$source.bl_occ_type";"=";"MEHandle")
```

Import or update a saved option set

If you want to import or update a saved option set, specify these closure rules in your transfer mode:

```
CLASS.Document:CLASS.VariantRule:CONTENT.*:DO:
CLASS.VariantRule:CLASS.*:CONTENT.*:DO:
CLASS.Document:CLASS.Option:CONTENT.*:DO:
CLASS.ProductRevision:CLASS.Option:ATTRIBUTE.optionRefs:DO:
CLASS.Option:CLASS.*:CONTENT.*:DO:
```

Add a filter list to optimize retrieval of BOM lines

You can limit the type of attachments to look for in a BOM line by creating a filter list using a closure rule. This increases the performance of PLM XML imports and exports. The form of the closure rule is as follows:

```
CLASS:PIESession:CLASS:CfgAttachmentWindow:PROPERTY:*:PROCESS
CLASS:PIESession:CLASS:CfgAttachmentWindow:PROPERTY:first—attachment—type:
   PROCESS
CLASS:PIESession:CLASS:CfgAttachmentWindow:PROPERTY:second—attachment—type:
   PROCESS
```

To define a filter list, the closure rule clauses must specify **PIESession** as the primary class and **CfgAttachmentWindow** as the secondary class. The first clause of the closure rule must specify ***** as the property so Teamcenter clears any previous filter list. Then add clauses with the attachment types you want to filter on. You can filter on one or more types. For example, if you want to see BOM lines with only renderings or specifications attached, write the following closure rule and add it to a transfer mode:

```
CLASS:PIESession:CLASS:CfgAttachmentWindow:PROPERTY:*:PROCESS
CLASS:PIESession:CLASS:CfgAttachmentWindow:PROPERTY:tc_rendering:PROCESS
CLASS:PIESession:CLASS:CfgAttachmentWindow:PROPERTY:
tc_specification:PROCESS
```

Use the bom_cb switch to avoid exporting the entire BOM

The Teamcenter PLM XML translator usually translates based on what the closure rule dictates as it performs the traversal. Occasionally, it deviates from this behavior.

For example, you cannot have a **ProductRevision** PLM XML entity without a **Product** entity. Therefore, if you export an **ItemRevision** entity from Teamcenter, you also get the related Teamcenter **Item** entity.

When you select a BOM line (or any configuration line) for export, the objects of interest are the selected line and all child lines. But the context of the selected line is lost without the top line and all the lines from the top line down to the selected line. So the translator moves back up the structure and exports all the BOM lines above the selected line, but it does not traverse the child lines of the lines above the selected lines.

Depending on your closure rules, you may find that you inadvertently get the child lines of the lines above, with the effect of always translating the full BOM. To avoid this, use the following clause:

```
TYPE.BOMLine:TYPE.*:PROPERTY.bl_all_child_lines:TRAVERSE+PROCESS:$bom_cb != “1”;
```

Skip ItemMaster forms on import

In PLM XML format, products may have several forms, even several **ItemMaster** forms. This is not the case in Teamcenter. If your import closure rule goes to an **ItemMaster** form or an **ItemRevisionMaster** form from some place other than the **Product** PLM XML entity and **ProductRevision** entity, the form is created but not correctly associated with the item or item revision. To avoid this, use the following clause:

```
CLASS.Document:CLASS.Form:CONTENT.*:SKIP
```

Place data in context on import

PLM XML supports editing in the context of an occurrence. Occurrences in a configuration view are always absolute, but you can indicate that the data in the file is intended to be altered in context or altered in a global scope. If the **UserData** tag under the occurrence contains the following line:

```
<UserValue value="TopLevel" title="AttributesContext"></UserValue>
```

the attributes, notes, and variant conditions under the occurrence are stored as an edit/create in the context of the top line. If this field is not found and the line has a context set for this data, the data is altered in that context. If there is no data, the data is altered globally. To support attachments in context, you can place a **UserData** tag in the **ProductView**, **PlantView**, or **ProcessView** tag's content. For example:

```
<ProductView id="id2" ruleRefs="#id1" rootRefs="id44" primaryOccurrenceRef="id4">

<UserData id="id136" type="AttributesInContext">

<UserValue value="TopLevel" title="AttributesContext"></UserValue></UserData>
```

The translator then interprets all new additions to be in the context of the top line.

Enable export of bounding box information

Add the following closure rule clause to enable the export of bounding box data:

```
TYPE.BOMLine:CLASS.BoundingBox:PROPERTY.bl_bounding_boxes:PROCESS /TRAVERSE+PROCESS
```

By default, bounding box data is not exported unless you add this clause to the closure rule.

When exported with this clause, a **Bound** object appears as a PLM XML element, typically under a **Representation** element. For example:

```
<Bound id="id17" type="compound”>
<Bound id="id18" values="1 2 3 4 5 6”></Bound>
<Bound id="id19" values="7 8 9 10 11 12"></Bound></Bound>
```

Closure rules for table properties

You can add, edit, or remove rows of table values after a Business Modeler IDE administrator adds a table property to a business object. This allows you to manage property data in a tabular format in the end-user interface.

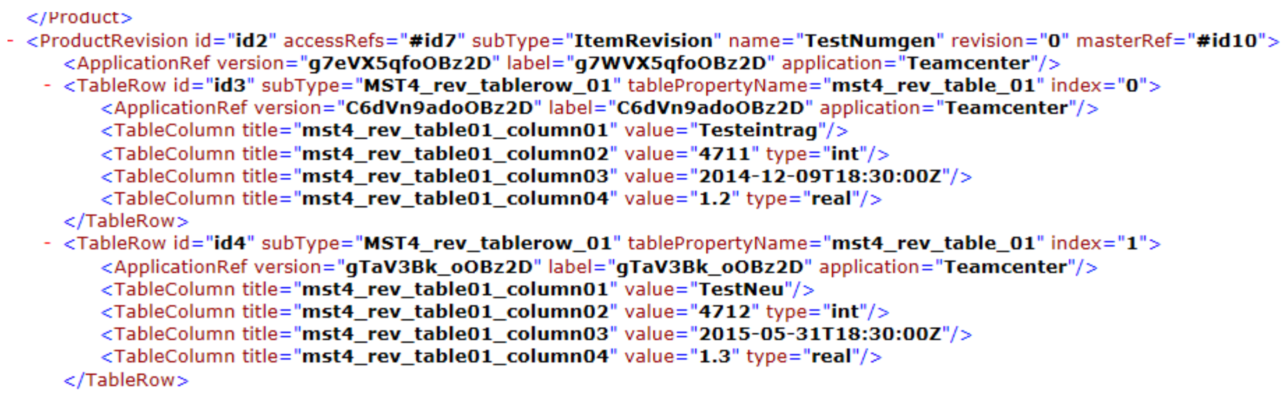
To export the values in a table, add the following closure rules for exporting table properties. This starts the table property export:

```
CLASS ItemRevision PROPERTY mst4_rev_table_01 DO
```

The following clauses trigger the columns that you want to export:

```
CLASS MST4_rev_tablerow_01 PROPERTY mst4_rev_table01_column01 DO
CLASS MST4_rev_tablerow_01 PROPERTY mst4_rev_table01_column02 DO
CLASS MST4_rev_tablerow_01 PROPERTY mst4_rev_table01_column03 DO
CLASS MST4_rev_tablerow_01 PROPERTY mst4_rev_table01_column04 DO
```

The export includes the table properties of the item revision:



```
  </Product>
- <ProductRevision id="id2" accessRefs="#id7" subType="ItemRevision" name="TestNumgen" revision="0" masterRef="#id10">
     <ApplicationRef version="g7eVX5qfoOBz2D" label="g7WVX5qfoOBz2D" application="Teamcenter"/>
   - <TableRow id="id3" subType="MST4_rev_tablerow_01" tablePropertyName="mst4_rev_table_01" index="0">
        <ApplicationRef version="C6dVn9adoOBz2D" label="C6dVn9adoOBz2D" application="Teamcenter"/>
        <TableColumn title="mst4_rev_table01_column01" value="Testeintrag"/>
        <TableColumn title="mst4_rev_table01_column02" value="4711" type="int"/>
        <TableColumn title="mst4_rev_table01_column03" value="2014-12-09T18:30:00Z"/>
        <TableColumn title="mst4_rev_table01_column04" value="1.2" type="real"/>
     </TableRow>
   - <TableRow id="id4" subType="MST4_rev_tablerow_01" tablePropertyName="mst4_rev_table_01" index="1">
        <ApplicationRef version="gTaV3Bk_oOBz2D" label="gTaV3Bk_oOBz2D" application="Teamcenter"/>
        <TableColumn title="mst4_rev_table01_column01" value="TestNeu"/>
        <TableColumn title="mst4_rev_table01_column02" value="4712" type="int"/>
        <TableColumn title="mst4_rev_table01_column03" value="2015-05-31T18:30:00Z"/>
        <TableColumn title="mst4_rev_table01_column04" value="1.3" type="real"/>
     </TableRow>
```

Closure rules for PublishLink

You can use PublishLink to connect absolute occurrences of source and target structures, such as connecting logically equivalent occurrences of design (source) and part (target) structures. However, you must add the following closure rules for PLM XML exporting and importing of PublishLink to work properly:

- Add the following closure rule clause to enable export of PublishLink data:

  ```
  Class.ImanItemLine:Class.PublishLink:Property.bl_is_publish_link_source
  :
     PROCESS+TRAVERSE
  ```

  By default, PublishLink data is not exported unless you add this clause to the closure rule.

- Add the following closure rule to enable import of PublishLink data:

  ```
  CLASS Document CLASS DefinitionRelation CONTENT * DO
  CLASS DefinitionRelation CLASS * CONTENT * DO
  CLASS DefinitionRelation CLASS * ATTRIBUTE * DO
  ```

After you add these closure rules, when a source (design) structure is exported, the PublishLink objects and the target (part) structures are also exported.

You can also use PublishLink to publish visualization-related data from the source to the target occurrence. To export visualization-related information, include rules that export that information. For example, when a shape is published from an occurrence of a component, its JT file is added to the target occurrence with the **IMANRendering** relation. If you want this information in the PLM XML file, you must add the appropriate closure rules for exporting this relation.

Teamcenter does not support the export of PublishLink data when exporting part structures.

Closure rules for part or design objects

To export and import part or design objects, add the following closure rules:

Export transfer mode

```
TYPE.PartRevision:TYPE.DesignRevision:RELATIONP2S:TC_Is_Represented_By:TRAVERSE_AND_PROCE
SS

    TYPE.DesignRevision:TYPE.PartRevision:RELATIONS2P:TC_Is_Represented_By:TRAVERSE_AND_P
    ROCESS
```

Import transfer mode

```
CLASS.Document:CLASS.RepresentedByRelation:CONTENT.*:DO
CLASS.RepresentedByRelation:CLASS.*:CONTENT.*:DO
CLASS.RepresentedByRelation:CLASS.*:ATTRIBUTE.*:DO
```

Exporting a GRM relation between items and item revisions

You can export a GRM relation between items and item revisions as follows:

- Relation between two items (**Item→Item**)
- Relation between two item revisions (**ItemRevision→ItemRevision**)
- Relation between an item and a revision of another item (**Item→ItemRevision**). For example, **Item1** is a relation of a revision of **Item2**.

Previously, PLM XML export had some instances where the **AssociatedDataSet** element was used to represent the relation. To match the Teamcenter data model, these scenarios have been changed to use the **GeneralRelation** element.

The following example shows that **RevA** of **Item1** has been related to **RevA** of **Item2** using a **FND_TraceLink** GRM.

Item1		Item2
RevA→	**FND_TraceLink**	→RevA

The closure rule clause to export this relation is:

```
CLASS ItemRevision CLASS ItemRevision RELATIONP2S * PROCESS+TRAVERSE
```

The exported **.xml** file looks similar to the following.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- GENERATED BY: PLM XML SDK 7.0.1.063 -->
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
 schemaVersion="6" date="2009-08-25" time="09:36:30"
 author="Teamcenter P8000.1.0.20090812.00 - infodba@tc810ms1(100001)">
<Header id="id1" traverseRootRefs="#id2"
 transferContext="ConfiguredDataExportDefault"></Header>
<ProductRevision id="id2" name="Item1" accessRefs="#id3" subType="ItemRevision"
 masterRef="#id6" revision="A">
<ApplicationRef version="EpApp4UIAABaaA" application="Teamcenter"
 label="EJFpp4UIAABaaA"></ApplicationRef></ProductRevision>
<ProductRevision id="id11" name="Item2" accessRefs="#id3"
 subType="ItemRevision" masterRef="#id14" revision="A">
<ApplicationRef version="ElOp5A34AABaaA" application="Teamcenter"
 label="EJDp5A34AABaaA"></ApplicationRef></ProductRevision>
<Product id="id6" name="Item1" accessRefs="#id3" subType="Item"
 productId="Item1">
<ApplicationRef version="EJFpp4UIAABaaA" application="Teamcenter"
 label="EJFpp4UIAABaaA"></ApplicationRef></Product>
<Product id="id14" name="Item2" accessRefs="#id3" subType="Item"
 productId="Item2">
<ApplicationRef version="EJDp5A34AABaaA" application="Teamcenter"
 label="EJDp5A34AABaaA"></ApplicationRef></Product>
<AccessIntent id="id3" intent="reference" ownerRefs="#id4">
</AccessIntent>
<Site id="id4" name="tc810ms1" siteId="123456789">
</Site>
<GeneralRelation id="id13" subType="FND TraceLink" relatedRefs="#id2 #id11">
<ApplicationRef version="E5Gp5oCSAABaaA" application="Teamcenter"
 label="E5Gp5oCSAABaaA"></ApplicationRef>
<UserData id="id12">
<UserValue value="Item1RevA->Item2RevA" title="name"></UserValue>
<UserValue value="Relation from Item1RevA to Item2RevA"
 title="description"></UserValue></UserData></GeneralRelation>
</PLMXML>
```

The highlighted items show that the **GeneralRelation** element relates **id2** and **id11**, representing the revisions for **Item1** and **Item2**, respectively.

Export forms and external files in a dataset

By default, forms and external files in a dataset are not exported as PLM XML elements. The following examples explain how to export forms and external files in a dataset.

Datasets are exported and stored in a peer directory of the exported XML file. The directory is named the same as the XML file's base name. **locationRef** attributes in the XML file's **ExternalFile** elements refer to the datasets in the directory using relative path references. This relative path and name relationship must be maintained to import the data. Do not change the directory name or attribute values if you modify data before importing it.

Export forms and external files

To export forms and external files in a dataset, add these clauses to the property set:

```
CLASS.Dataset:Property.ref_names:DO:
```

```
CLASS.Dataset:Property.ref_list:DO:
```

Note:

For the forms and external files saved from NX, alternately, the NX **ug_clone** utility may be used by setting the **PIE_IMF_FOR_CAD** preference to a value of **NO** in PLM XML export.

In this case, all forms and external files in the dataset appear as a PLM XML element, and the related **AssociatedForm** elements are added under the **DataSet** element.

```
<DataSet id="id8" name="test0917/A" accessRefs="#id4" version="1"
  memberRefs="#id12" type="CAEMesh">
<ApplicationRef version="wqbRBu28AABaaA" application="Teamcenter"
label="wqbRBu28AABaaA"/>
<AssociatedForm id="id11" role="ref_list" formRef="#id9"/>
<UserData id="id15">
<UserValue value="" title="ref_names">
<UserList id="id13" type="list">
<Item value="UGPART-ATTR"/>
<Item value="FEM_Mesh"/>
</UserList>
</UserValue>
<UserValue value="" title="ref_list">
<UserList id="id14" type="list">
<Item value="id9"/>
<Item value="id12"/>
</UserList>
</UserValue>
</UserData>
</DataSet>
<Form id="id9" name="UGPART-ATTR" accessRefs="#id4" subType="UGPartAttr"
  subClass="UGPartAttr">
<Description>This is a form for UGPART</Description>
<ApplicationRef version="wuZRBu28AABaaA" application="Teamcenter"
label="wuZRBu28AABaaA"/>
```

```
<UserData id="id10" type="FormAttributes">
<UserValue value="" title="assemb_no"/>
<UserValue value="" title="creator"/>
<UserValue value="" title="material"/>
<UserValue value="" title="model"/>
<UserValue value="" title="part_no"/>
<UserValue value="" title="part_type"/>
<UserValue value="" title="task_no"/>
<UserValue value="" title="title"/>
<UserValue type="real" value="" title="weight"/>
</UserData>
</Form>
<ExternalFile id="id12" accessRefs="#id4" locationRef="fixed\cad001.dat" format="dat">
<ApplicationRef version="AXfRBu28AABaaA" application="Teamcenter"
label="AXfRBu28AABaaA"/>
</ExternalFile>
```

Export forms only

To export only forms in a dataset, add this clause to the closure rule and do not use the property set clause described in the previous example.

```
CLASS.Dataset:CALSS.Form:ATTRIBUTE.ref_list:TRAVERSE_AND_PROCESS
```

When exported with this clause, all forms in the dataset appear as PLM XML elements. Related **AssociatedForm** elements are added under **DataSet** element. For example:

```
<DataSet id="id9" name="J60621-A" accessRefs="#id10" version="1" type="UGMASTER">
<ApplicationRef version="xFfNnDPlAABaaA" application="Teamcenter"
label="xFfNnDPlAABaaA"/>
<AssociatedForm id="id15" role="ref_list" formRef="#id13"/>
<AssociatedForm id="id18" role="ref_list" formRef="#id16"/>
<AssociatedForm id="id24" role="ref_list" formRef="#id19"/>
<AssociatedForm id="id30" role="ref_list" formRef="#id28"/>
</DataSet>
<Form id="id13" name="UGPART-ATTR" accessRefs="#id10" subType="UGPartAttr"
  subClass="UGPartAttr">
<Description>This is a form for UGPART</Description>
<ApplicationRef version="xJaNnDPlAABaaA" application="Teamcenter"
label="xJaNnDPlAABaaA"/>
<UserData id="id14" type="FormAttributes">
<UserValue value="" title="assemb_no"/>
<UserValue value="" title="creator"/>
<UserValue value="" title="material"/>
<UserValue value="" title="model"/>
<UserValue value="" title="part_no"/>
<UserValue value="" title="part_type"/>
<UserValue value="" title="task_no"/>
<UserValue value="" title="title"/>
<UserValue type="real" value="" title="weight"/>
</UserData>
</Form>
```

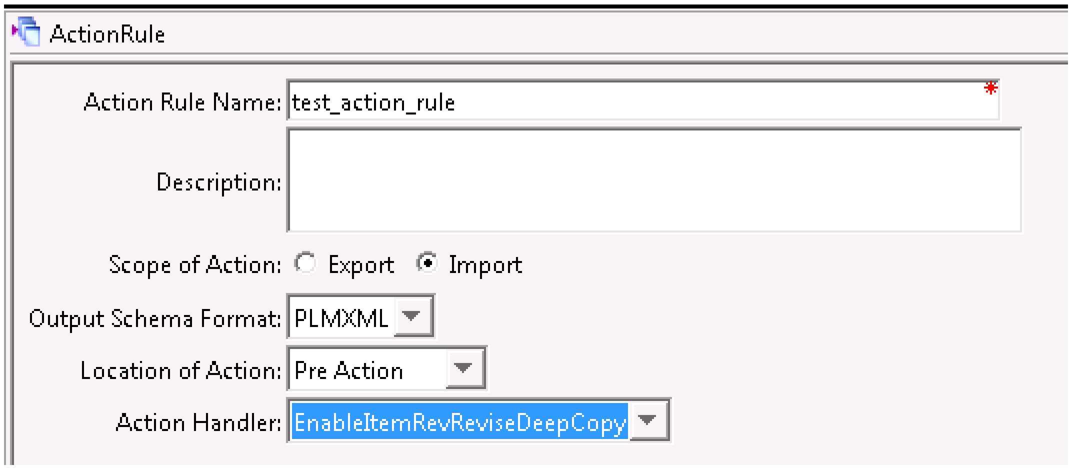
Enable item revision deep copy import

A PLM XML transfer mode can be configured to revise an item revision during a PLM XML import. The changed item revision contains objects from the previous item revision because of Teamcenter deep copy rules. The configuration includes a new action rule that uses the COTS **EnableItemRevReviseDeepCopy** action handler.

The changed behavior of the BOM view revision under the item revision is not controlled by the action handler. Typically, this action handler is used to copy the previous item revision's attachments and to handle relationships to the new item revision during the revise operation.

By default, PLM XML import only creates objects defined within the PLM XML file; therefore, the PLM XML file must only contain PLM XML elements for an item revision, specifically **ProductRevision** (item revision) and **Product** (item).

1. In PLM XML/TC XML Export Import Administration, create a new action rule that uses the **EnableItemRevReviseDeepCopy** action handler.



Action Rule Name: test_action_rule

Description:

Scope of Action: Export [✓ Import]

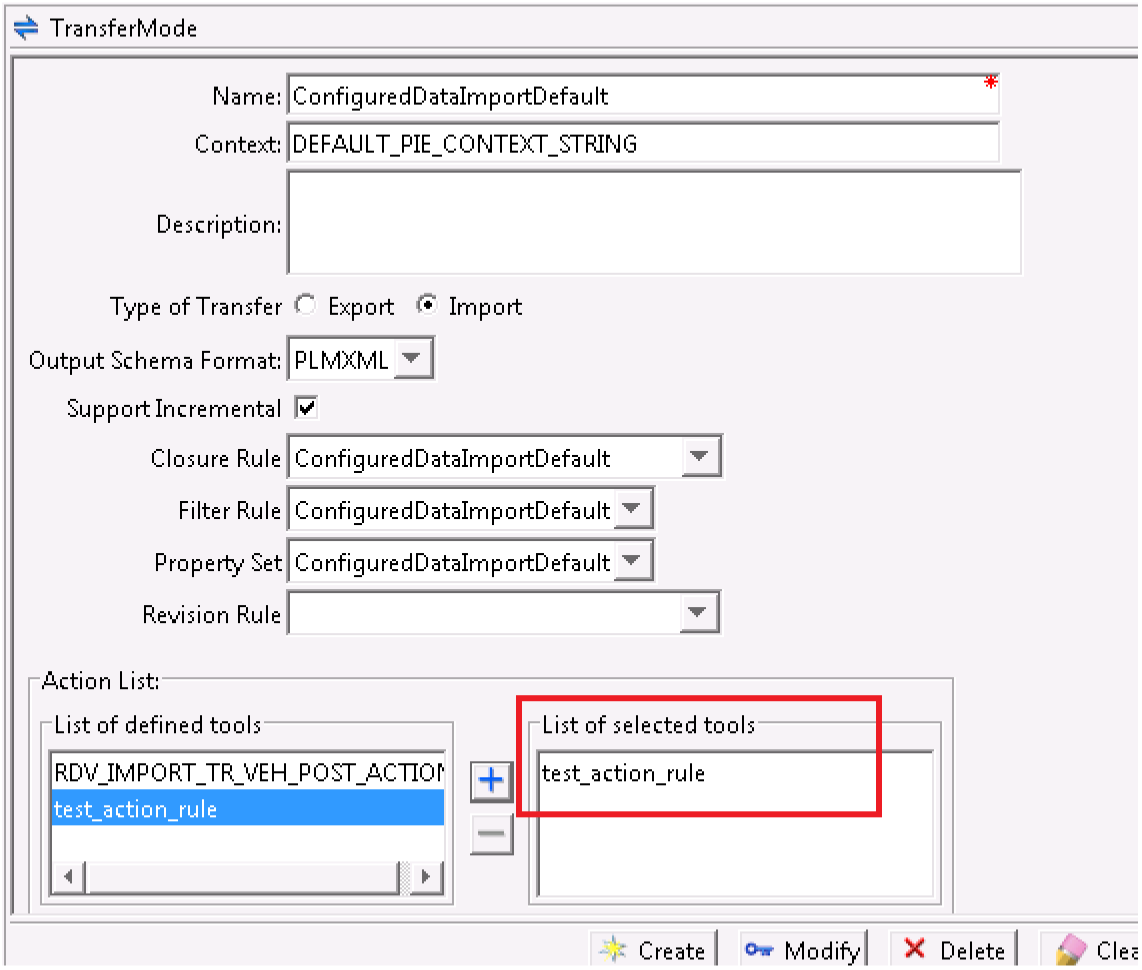
Output Schema Format: PLMXML

Location of Action: Pre Action

Action Handler: EnableItemRevReviseDeepCopy

Location of Action can be **Pre Action** or **During Action** but not **Post Action**.

2. Modify an existing transfer mode or create a new transfer mode. Add the new action handler to the **List of selected tools** list.


TransferMode

Name: ConfiguredDataImportDefault

Context: DEFAULT_PIE_CONTEXT_STRING

Description:

Type of Transfer ○ Export [✓ Import]

Output Schema Format: PLMXML

Support Incremental ✓

Closure Rule ConfiguredDataImportDefault

Filter Rule ConfiguredDataImportDefault

Property Set ConfiguredDataImportDefault

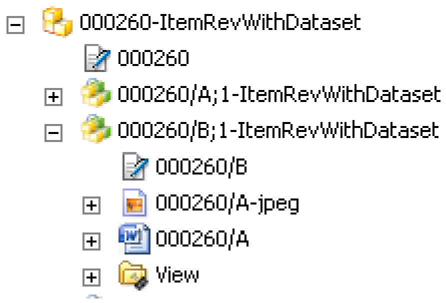
Revision Rule

Action List:

List of defined tools: RDV_IMPORT_TR_VEH_POST_ACTIOI, test_action_rule

List of selected tools: test_action_rule

3. Before the PLM XML import, the **000260** item contains two revisions with attached objects as shown.


```
000260-ItemRevWithDataset
  000260
  000260/A;1-ItemRevWithDataset
  000260/B;1-ItemRevWithDataset
    000260/B
    000260/A-jpeg
    000260/A
    View
```

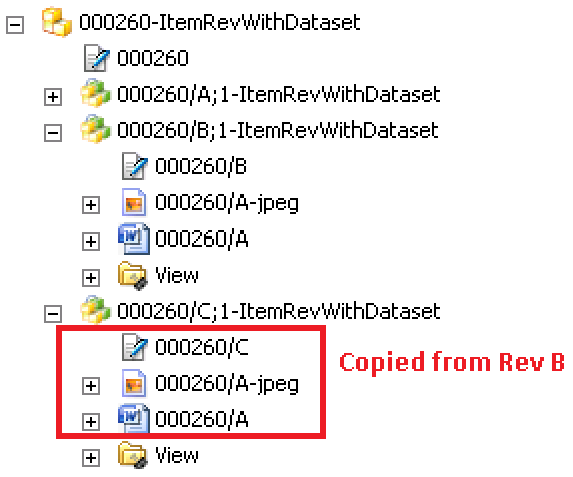
4. Prepare a PLM XML file that only contains **ProductRevision** (item revision) and **Product** (item) elements. Ensure the PLM XML file does not contain any **ApplicationRef** elements or elements for a dataset or form that is attached to the item revision

```
<?xml version="1.0" encoding="utf-8"?>
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="6" language="en-us" date="2021-10-06" time="15:58:36"
  author="Teamcenter V10000.1.0.20_20210617.00 - tcdba@IMC--1484631002(-1484631002)">
```

```
<Header id="id1" traverseRootRefs="#id2"
transferContext="ConfiguredDataExportDefault" />
  </Header>
<ProductRevision id="id2" name="ItemRevWithDataset" accessRefs="#id4"
  subType="ItemRevision" masterRef="#id7"></ProductRevision>
<Product id="id7" name="ItemRevWithDataset" accessRefs="#id4" subType="Item"
  productId="000260" /></Product>
</PLMXML>
```

5. Import the PLM XML file using the transfer mode that contains the new action handler.

   Teamcenter creates the new item revision from the PLM XML file and applies deep copy rules on all the objects of the latest existing revision to the new item revision.



```
000260-ItemRevWithDataset
  000260
  000260/A;1-ItemRevWithDataset
  000260/B;1-ItemRevWithDataset
    000260/B
    000260/A-jpeg
    000260/A
    View
  000260/C;1-ItemRevWithDataset
    000260/C
    000260/A-jpeg
    000260/A
    View
```

Exporting owning group information

You can create a closure rule and property set to export the owning group for various types of objects. For example, using the BMIDE, create a compound (or other) property that points to the **owning_group** object's name.

Be aware that the output includes all roles and volumes associated with that group. In some cases, this additional information is not appropriate due to sensitivity issues.

To export the owning group name without all the roles associated with the group, create an owning group property on the item revision (as a string) and export that property only.

Exporting folder contents

To export the contents of a folder, use the following closure rule:

```
CLASS.Folder:CLASS.*:PROPERTY.contents:TRAVERSE+PROCESS
```

To export only the items in a folder, use the following closure rule:

```
CLASS.Folder:CLASS.Item:PROPERTY.contents:TRAVERSE+PROCESS
```

Closure rules for incremental change exports

When creating a custom incremental change (IC) transfer mode, you must include the following closure rules as a minimum to ensure a complete and correct IC-based delta export:

```
CLASS.ItemRevision:CLASS.ReleaseStatus:ATTRIBUTE.release_status_list:TRAVERSE+PROCESS

TYPE.EngChangeRevision:CLASS.ReleaseStatus:ATTRIBUTE.release_status_list:TRAVERSE+PROCESS

CLASS.ReleaseStatus:CLASS.Effectivity:ATTRIBUTE.effectivities:TRAVERSE+PROCESS

TYPE.EngChangeRevision:CLASS.Form:RELATIONP2S.IMAN_specification:PROCESS

TYPE.EngChangeRevision:CLASS.Form:RELATIONP2S.IMAN_master_form:PROCESS
```

Export EPMTask and Signoff object properties

A PLM XML export transfer mode can export **EPMTask** and **Signoff** objects with the following properties as attributes, if the values exist:

EPMTask

responsiblePartyRef
activeSurrogateRef
assignerRef
userRef (performer)

Signoff

activeSurrogateRef
assignerRef
userRef (performer)

Create the transfer mode using these closure rule clauses:

```
CLASS.ItemRevision:CLASS.EPMTask:PROPERTY.process_stage_list:TRAVERSE+PRO
CESS

CLASS.EPMTask:CLASS.Signoff:PROPERTY.signoff_attachments:TRAVERSE+PROCESS

CLASS.Signoff:CLASS.GroupMember:PROPERTY.active_surrogate:TRAVERSE+PROCES
S
```

Create the transfer mode using these property set clauses:

```
CLASS.Signoff:Property.object_name:DO:
```

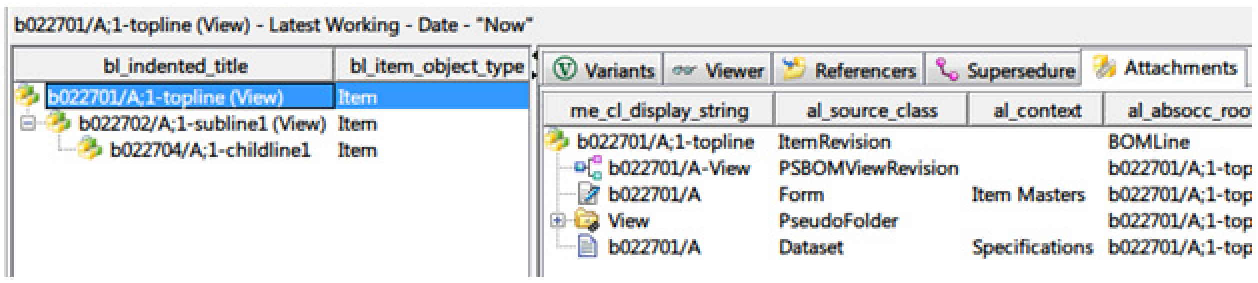
```
CLASS.Signoff:Property.signoff_type:DO:

CLASS.Signoff:Property.fnd0Assignee:DO:

CLASS.EPMTask:Property.object_string:DO:

CLASS.EPMTask:Property.object_name:DO:

CLASS.EPMTask:Property.root_target_attachments:DO:

CLASS.EPMTask:Property.fnd0Assignee:DO:
```

Creating a closure rule using an example

The following series of topics explains the process of building a custom PLM XML closure rule to traverse a BOM structure. The steps include exporting the various parts of a BOM, including the BOM window top line, child lines, attachments, datasets, and item revisions. The default closure rule **ConfiguredDataExportDefault** may not be an ideal choice if you are traversing a large structure as it traverses through many objects and relations.

This example assumes you have basic Teamcenter data modeling knowledge of how a BOM structure is constructed and the underlying relations between Teamcenter objects.

Following is a BOM structure in Structure Manager, where some of the BOM lines also have datasets.



b022701/A;1-topline (View) - Latest Working - Date - "Now"

bl_indented_title	bl_item_object_type
b022701/A;1-topline (View)	Item
b022702/A;1-subline1 (View)	Item
b022704/A;1-childline1	Item

me_cl_display_string	al_source_class	al_context	al_absocc_roo
b022701/A;1-topline	ItemRevision		BOMLine
b022701/A-View	PSBOMViewRevision		b022701/A;1-top
b022701/A	Form	Item Masters	b022701/A;1-top
View	PseudoFolder		b022701/A;1-top
b022701/A	Dataset	Specifications	b022701/A;1-top

Specify only one structure to export at a time. This is applicable for the export from the Teamcenter rich client, the **plmxml_export** utility, and as the **PIE_session_export_objects** ITK function.

Step 1: Create session rules

When you export data, it is placed in the **target_list** property of the PIESession object. PIESession is a runtime object that holds the entire session information of an export or import.

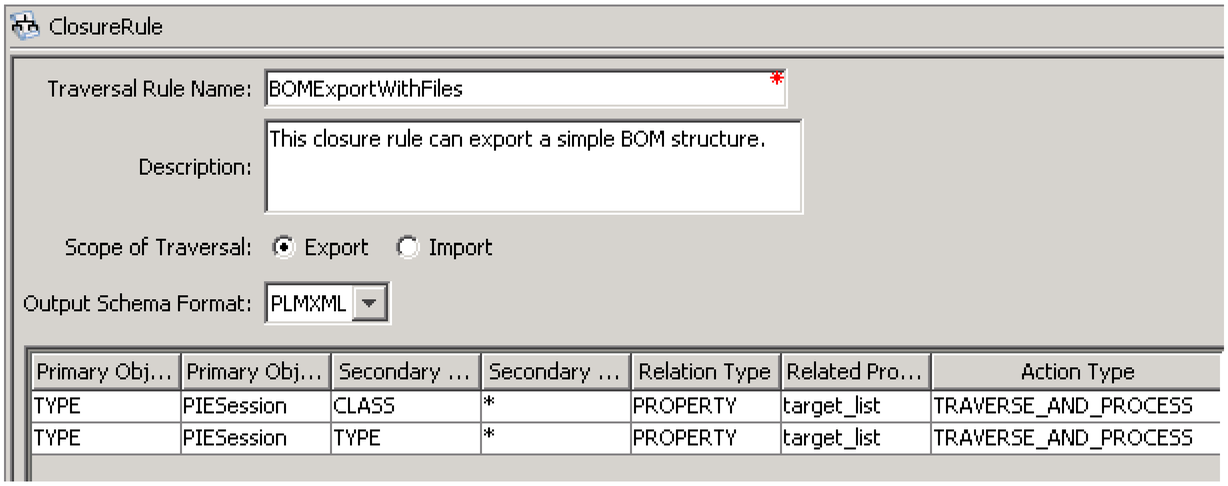
To ensure that all target objects set to the PIESession are processed and traversed, create the following closure rules. For step-by-step instructions, see *Create closure rules*.

```
TYPE.PIESession:CLASS.* : PROPERTY.target_list : TRAVERSE_AND_PROCESS:
TYPE.PIESession:TYPE.* : PROPERTY.target_list : TRAVERSE_AND_PROCESS:
```

Tip:

The **TRAVERSE** and **PROCESS** actions can be combined by using the shift key.

This example shows a simple closure rule with these two clauses.



Traversal Rule Name:	BOMExportWithFiles
Description:	This closure rule can export a simple BOM structure.
Scope of Traversal:	[✓ Export] Import
Output Schema Format:	PLMXML

Primary Obj…	Primary Obj…	Secondary …	Secondary …	Relation Type	Related Pro…	Action Type
TYPE	PIESession	CLASS	*	PROPERTY	target_list	TRAVERSE_AND_PROCESS
TYPE	PIESession	TYPE	*	PROPERTY	target_list	TRAVERSE_AND_PROCESS

Step 2: Create a traversal rule to export the BOM window top line

Once you have the session rules created, you begin creating the closure rules to export the BOM lines.

To traverse the top line by the **top_line** property of BOM window, create the following closure rule:

```
TYPE.BOMWindow:TYPE.*:PROPERTY.top_line:TRAVERSE_AND_PROCESS:
```

With this closure rule, the export includes the BOM window and top line:

```
<ProductView id="id4" ruleRefs="#id2" rootRefs="id6"
    primaryOccurrenceRef="id6">
...
<Occurrence id="id6">
...
    </Occurrence>
    </ProductView>
```

Step 3: Create traversal rules to export child lines of a BOM line

1. Traverse child lines by using the **bl_all_child_lines** property of parent lines:

```
CLASS.ImanItemLine :
TYPE.* :PROPERTY.bl_all_child_lines:TRAVERSE_AND_PROCESS:
```

2. Add a conditional clause for two specific cases:

   - To prevent this rule from applying to the BOP window, add this:

     ```
     PRIMARY.bl_window_is_BOPWin=="false"
     ```

   - When an intermediate line in a BOM structure is selected for export, you export the entire subassembly below the selected line and parent lines all the way to the top (referred to as *BOM crawl back*). This provides the context of the BOM line with respect to the parent or the top line. During crawl back, traversing to the children exports the siblings of the selected line, resulting in a full BOM export.

     Add the following to prevent a full BOM export:

     ```
     $bom_cb != "1"
     ```

   Doing so ensures that the child lines are traversed only when the export is not in crawl back mode. The **bom_cb** option is set to **1** internally by the PLM XML framework whenever the traversal is backward.

The entire closure rule clause appears as follows:

```
CLASS.ImanItemLine :
TYPE.* :PROPERTY.bl_all_child_lines:TRAVERSE_AND_PROCESS:
    $bom_cb != "1" && PRIMARY.bl_window_is_BOPWin=="false"
```

With this closure rule, the export includes the BOM window and all BOM lines. Each occurrence element has **occurrenceRefs** pointing to its child lines.

```
<ProductView id="id4" ruleRefs="#id2" rootRefs="id6"
    primaryOccurrenceRef="id6">
...
<Occurrence id="id6" occurrenceRefs="id9">
...
    </Occurrence>
<Occurrence id="id9" occurrenceRefs="id12" parentRef="#id6">
...
    </Occurrence>
<Occurrence id="id12" parentRef="#id9">
...
    </Occurrence>
    </ProductView>
```

Step 4: Create traversal rules to export attachments of a BOM line

1. Use the **bl_attachments** property as follows to export the attachments of a BOM line such as forms and datasets:

   ```
   CLASS.ImanItemLine : TYPE.* :
   PROPERTY.bl_attachments:TRAVERSE_AND_PROCESS:
   ```

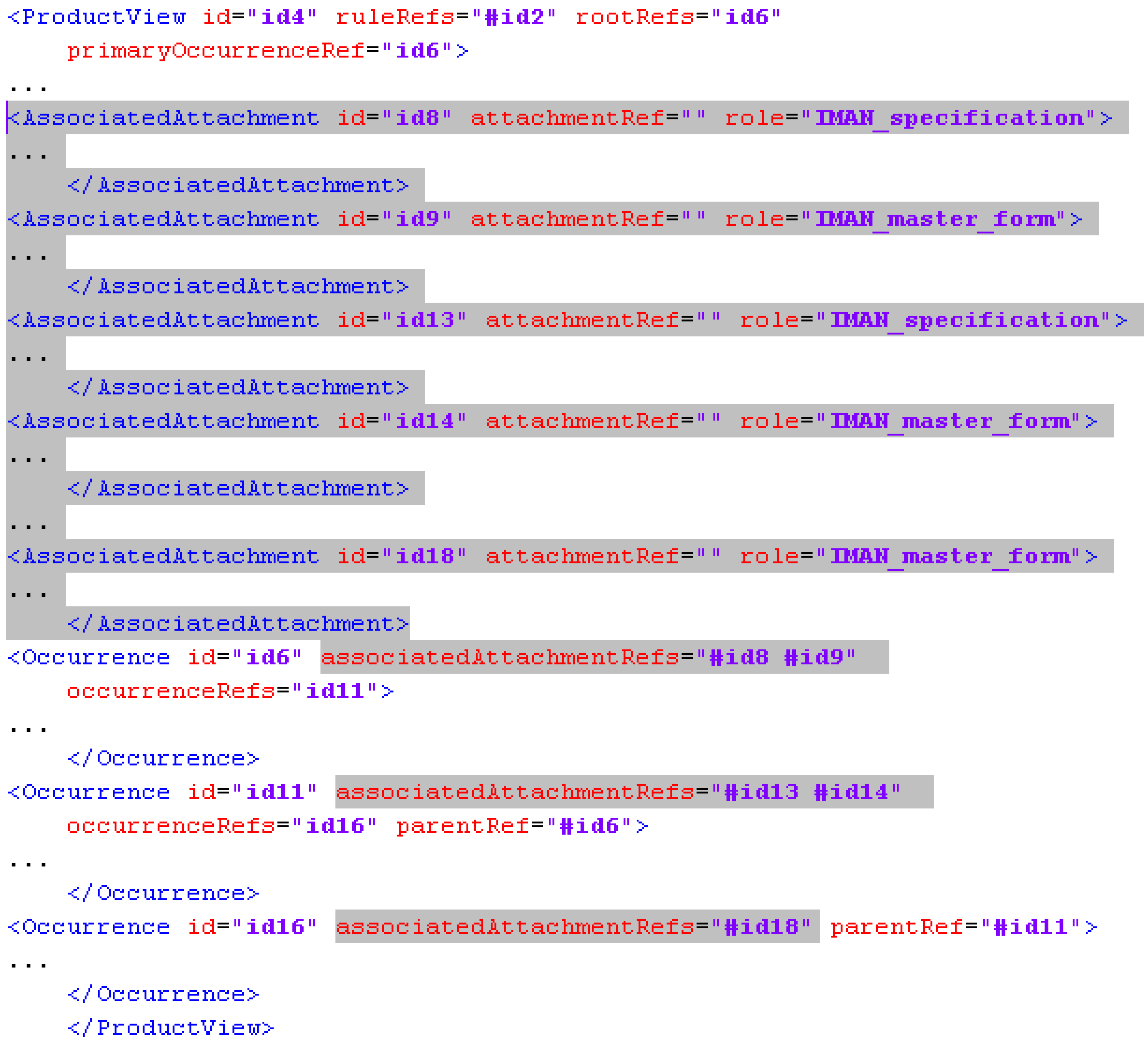
2. To limit the attachments to the **Dataset**, **Form**, and **Folder** types and exclude the BOP window cases, add these clauses:

   ```
   SECONDARY.al_source_class=="Dataset"
   SECONDARY.al_source_class=="Form"
   SECONDARY.al_source_class=="Folder"
   PRIMARY.bl_window_is_BOPWin=="false"
   ```

The entire closure rule appears as follows:

```
CLASS.ImanItemLine : TYPE.* :
PROPERTY.bl_attachments:TRAVERSE_AND_PROCESS:
    SECONDARY.al_source_class=="Dataset" &&
PRIMARY.bl_window_is_BOPWin=="false"
CLASS.ImanItemLine : TYPE.* :
PROPERTY.bl_attachments:TRAVERSE_AND_PROCESS:
    SECONDARY.al_source_class=="Form" &&
PRIMARY.bl_window_is_BOPWin=="false"
CLASS.ImanItemLine : TYPE.* :
PROPERTY.bl_attachments:TRAVERSE_AND_PROCESS:
    SECONDARY.al_source_class=="Folder" &&
PRIMARY.bl_window_is_BOPWin=="false"
```

With this closure rule, the export includes the attachments. The attachment references for each occurrence point to **AssociatedAttachment** element. These elements then point to actual attachment object using **attachmentRef** (form or dataset).



```
<ProductView id="id4" ruleRefs="#id2" rootRefs="id6"
    primaryOccurrenceRef="id6">
...
<AssociatedAttachment id="id8" attachmentRef="" role="IMAN_specification">
...
    </AssociatedAttachment>
<AssociatedAttachment id="id9" attachmentRef="" role="IMAN_master_form">
...
    </AssociatedAttachment>
<AssociatedAttachment id="id13" attachmentRef="" role="IMAN_specification">
...
    </AssociatedAttachment>
<AssociatedAttachment id="id14" attachmentRef="" role="IMAN_master_form">
...
    </AssociatedAttachment>
...
<AssociatedAttachment id="id18" attachmentRef="" role="IMAN_master_form">
...
    </AssociatedAttachment>
<Occurrence id="id6" associatedAttachmentRefs="#id8 #id9"
    occurrenceRefs="id11">
...
    </Occurrence>
<Occurrence id="id11" associatedAttachmentRefs="#id13 #id14"
    occurrenceRefs="id16" parentRef="#id6">
...
    </Occurrence>
<Occurrence id="id16" associatedAttachmentRefs="#id18" parentRef="#id11">
...
    </Occurrence>
    </ProductView>
```

Step 5: Create traversal rules to export child attachment lines

An attachment line can be attached to another attachment line as a child line.

To include all attachment lines, use a closure rule such as the following:

```
TYPE.CfgAttachmentLine : TYPE.CfgAttachmentLine :
PROPERTY.me_cl_child_lines:
    TRAVERSE_AND_PROCESS:((SECONDARY.al_source_class!="PseudoFolder") &&
    (SECONDARY.al_context!=    "IMAN_PublishingPageAssetContext"))
```

This closure rule includes two conditions for the following purposes:

- To prevent traversing child lines under a **PseudoFolder**, as it is a runtime object that cannot be used for traversal.

- To prevent child attachment lines with the relation **IMAN_PublishingPageAssetContext**. Child attachment lines may introduce circular links, causing infinite traversal loops. Specifying the relation in the conditional clause prevents circular links.

Step 6: Create traversal rules to export the actual attachment object

After you have created the closure rules to export attachment lines, you can create the closure rules to export the actual attachment objects.

Use the **me_cl_source** property to include the actual attachment object, such as a dataset or form, in the export from an attachment line, as follows:

```
TYPE.CfgAttachmentLine : CLASS.Dataset:
PROPERTY.me_cl_source : TRAVERSE_AND_PROCESS:
```

Step 7: Create traversal rules to export dataset named references (ImanFile)

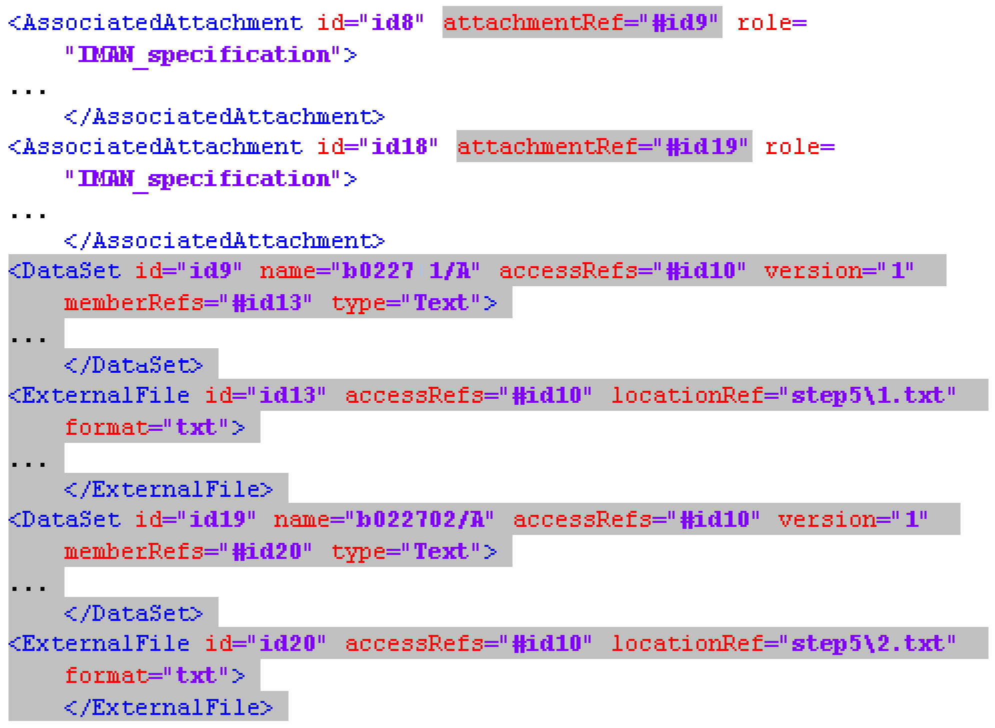
Use these two clauses to export dataset named references that are ImanFiles:

```
CLASS.Dataset:CLASS.ImanFile:ATTRIBUTE.ref_list:TRAVERSE_AND_PROCESS:
CLASS.ImanFile:CLASS.*:ATTRIBUTE.original_file_name:PROCESS:
```

The first clause traverses the ImanFile named references of a dataset. In some cases, the named references can point to forms and other types, but they cannot be traversed using the clause. You can use CLASS * to traverse to all named references of a dataset; however, the export result depends on whether or not the underlying named reference is supported in the PLM XML schema.

The second clause triggers the actual physical file export from a volume to the export directory:

With these traversal rules, the related dataset is exported:



```
<AssociatedAttachment id="id8" attachmentRef="#id9" role=
    "IMAN_specification">
...
    </AssociatedAttachment>
<AssociatedAttachment id="id18" attachmentRef="#id19" role=
    "IMAN_specification">
...
    </AssociatedAttachment>
<DataSet id="id9" name="b0227 1/A" accessRefs="#id10" version="1"
    memberRefs="#id13" type="Text">
...
    </DataSet>
<ExternalFile id="id13" accessRefs="#id10" locationRef="step5\1.txt"
    format="txt">
...
    </ExternalFile>
<DataSet id="id19" name="b022702/A" accessRefs="#id10" version="1"
    memberRefs="#id20" type="Text">
...
    </DataSet>
<ExternalFile id="id20" accessRefs="#id10" locationRef="step5\2.txt"
    format="txt">
    </ExternalFile>
```

Step 8: Create a traversal rule to traverse to an item revision

Export the underlying configured revision for each BOM line of the assembly using the following clause:

```
CLASS.ImanItemLine:TYPE.*:PROPERTY.bl_revision:
TRAVERSE_AND_PROCESS: PRIMARY.bl_window_is_BOPWin=="false"
```

The condition clause at the end prevents this rule from applying to the BOP window. The export includes the item revision related to the BOM line. The item is also exported, as the item revision must be included with the item.

```
<ProductRevision id="id26" name="childline1" accessRefs="#id10" subType=
    "ItemRevision" masterRef="#id27" revision="A">
...
    </ProductRevision>
<ProductRevision id="id28" name="subline1" accessRefs="#id1 " subType=
    "ItemRevisi n" masterRef="#id29" revision="A">
...
    </ProductRevision>
<ProductRevision id="id30" name="topline" accessRefs="#id10" subType=
    "ItemRevision" masterRef="#id31" revision="A">
...
    </ProductRevision>
<Product id="id27" name="childline1" accessRefs="#id10" subType="Item"
    productId="b022704">
...
    </Product>
<Product id="id29" name="subline1" accessRefs="#id10" subType="Item"
    productId="b022702">
...
    </Product>
<Product id="id31" name="topline" accessRefs="#id10" subType="Item"
    productId="b022701">
...
    </Product>
...
<Occurrence id="id6" instancedRef="#id30" associatedAttachmentRefs=
    "#id8 #id14" occurrenceRefs="id16">
...
    </Occurrence>
<Occurrence id="id16" instancedRef="#id28" associatedAttachmentRefs=
    "#id18 #id21" occurrenceRefs="id23" parentRef="#id6">
...
    </Occurrence>
<Occurrence id="id23" instancedRef="#id26" associatedAttachmentRefs=
    "#id25" parentRef="#id16">
...
    </Occurrence>
```

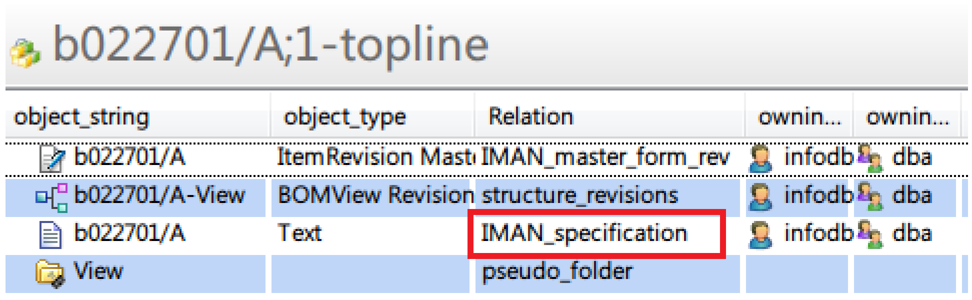
Step 9: Create a traversal rule to traverse to an item from an item revision

Use this closure rule to explicitly specify that an item be exported when its item revision is exported:

```
CLASS.ItemRevision:CLASS.Item:PROPERTY.items_tag:TRAVERSE_AND_PROCESS:
```

Step 10: Create a traversal rule to traverse to a dataset from an item revision

1. When a dataset is added to an item revision, define one kind of relationship; for example, if you add a dataset, the default relationship is the IMAN specification.



b022701/A;1-topline

object_string	object_type	Relation	ownin...	ownin...
b022701/A	ItemRevision Mast	IMAN_master_form_rev	infodb	dba
b022701/A-View	BOMView Revision	structure_revisions	infodb	dba
b022701/A	Text	IMAN_specification	infodb	dba
View		pseudo_folder		

2. Traverse the dataset by specifying the relation in a closure rule:

```
CLASS.ItemRevision:CLASS.Dataset:PROPERTY.IMAN_specification:
TRAVERSE_AND_PROCESS:
```

The export includes the relation between the item revision and the dataset:

```
<ProductRevision id="id28" name="subline1" accessRefs="#id10" subType=
    "ItemRevision" masterRef="#id29" revision="A">
...
<AssociatedDataSet id="id30" dataSetRef="#id19" role="IMAN_specification">
...
    </AssociatedDataSet>
    </ProductRevision>
<ProductRevision id="id31" name="topline" accessRefs="#id10" subType=
    "ItemRevision" masterRef="#id32" revision="A">
...
<AssociatedDataSet id="id33" dataSetRef="#id9" role="IMAN_specification">
...
    </AssociatedDataSet>
    </ProductRevision>
```

Step 11: Review the closure rule example file

After you combine all of the clauses in the preceding steps, ensure the closure rule appears as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<TCXML xmlns="http://www.tcxml.org/Schemas/TCXMLSchema" format="high_level">
<ClosureRule name="BOMExportWithFiles"
             clauses="TYPE.PIESession:CLASS.* : PROPERTY.target_list :
TRAVERSE_AND_PROCESS:,
                     TYPE.PIESession:TYPE.* : PROPERTY.target_list :
TRAVERSE_AND_PROCESS:,
                     TYPE.BOMWindow:TYPE.*:PROPERTY.top_line:PROCESS+TRAVERSE:,
                     CLASS.ImanItemLine :TYPE.*:PROPERTY.bl_all_child_lines:
PROCESS+TRAVERSE: $bom_cb != &quot;1&quot; &amp;&amp;
PRIMARY.bl_window_is_BOPWin==&quot;false&quot;,
                     CLASS.ImanItemLine :TYPE.*:PROPERTY.bl_attachments:
PROCESS+TRAVERSE:SECONDARY.al_source_class==&quot;Dataset&quot;
&amp;&amp; PRIMARY.bl_window_is_BOPWin==&quot;false&quot;,
                     CLASS.ImanItemLine :TYPE.*:PROPERTY.bl_attachments:
PROCESS+TRAVERSE:SECONDARY.al_source_class==&quot;Form&quot;
```

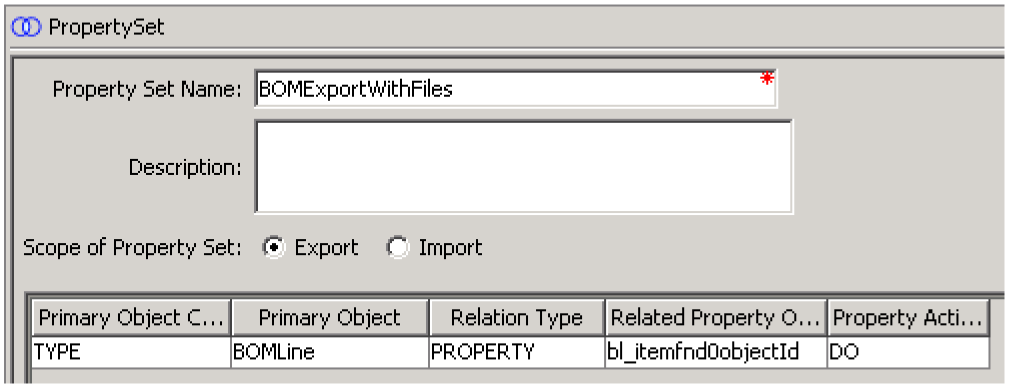
```
&amp;&amp; PRIMARY.bl_window_is_BOPWin==&quot;false&quot;,
                     CLASS.ImanItemLine :TYPE.*:PROPERTY.bl_attachments:
PROCESS+TRAVERSE:SECONDARY.al_source_class==&quot;Folder&quot;
&amp;&amp;PRIMARY.bl_window_is_BOPWin==&quot;false&quot;,
                     TYPE.CfgAttachmentLine:TYPE.CfgAttachmentLine:
PROPERTY.me_cl_child_lines:PROCESS+TRAVERSE:
((SECONDARY.al_source_class!=&quot;PseudoFolder&quot;)
&amp;&amp;(SECONDARY.al_context!=&quot;IMAN_PublishingPageAssetContext&quot;)),
             TYPE.CfgAttachmentLine:CLASS.Dataset:PROPERTY.me_cl_source:
PROCESS+TRAVERSE:,
             CLASS.Dataset:CLASS.ImanFile:ATTRIBUTE.ref_list:PROCESS+TRAVERSE:,
             CLASS.ImanFile:CLASS.*:ATTRIBUTE.original_file_name:PROCESS:,
             CLASS.ImanItemLine:TYPE.*:PROPERTY.bl_revision:PROCESS+TRAVERSE:,
             CLASS.ItemRevision:CLASS.Item:PROPERTY.items_tag:PROCESS+TRAVERSE:,
             CLASS.ItemRevision:CLASS.Dataset:PROPERTY.IMAN_specification:
PROCESS+TRAVERSE: "
description="This closure rule can export a simple BOM structure with
corresponding Item, Rev, Datasets, Forms, Files and BOMLine attachments."
             elemId="id6"
             owning_site="#id2"
             schema_format="0"
             scope="0">
  <GSIdentity elemId="id5"
             label="Q2H90lAIAABaaA"/>
</ClosureRule>

<Group elemId="id8"
       list_of_role="#id7"
       name="dba"
       owning_site="#id2">
  <GSIdentity elemId="id4"
            label="AkH9DZ$9AABaaA"/>
</Group>

<POM_imc elemId="id9"
         name="tc10ms1"
         owning_site="#id2"
         site_id="100001">
  <GSIdentity elemId="id2"
              label="w9F9DZeHAABaaA"/>
</POM_imc>

<TransferMode object_name="BOMExportWithFiles"
              object_desc="This TransferMode can export a simple BOM structure
with corresponding Item, Rev, Datasets, Forms, Files and BOMLine attachments."
              context_string="DEFAULT_PIE_CONTEXT_STRING"
              direction="0"
              closure_rules="#id5"
              filters=""
              action_list=""
              prop_sets=""
              config_objs=""
              schema_format="0"
              incremental="Y"
              multi_site="N"
              elemId="id10"
              gov_classification=""
              ip_classification=""
              license_list=""
              owning_site="#id2"
              project_list=""  >
  <GSIdentity elemId="id1"
              label="gDH90qy0AABaaA"/>
</TransferMode>

<User elemId="id11"
      owning_site="#id2"
      user_id="tcdba">
  <GSIdentity elemId="id3"
              label="AsL9DZ$9AABaaA"/>
</User>

<Role elemId="id12"
      owning_site="#id2"
      role_name="DBA">
  <GSIdentity elemId="id7"
              label="AgD9DZ$9AABaaA"/>
</Role>

<Header author="tcdba"
        date="2021-02-27"
        elemId="id13"
        originatingSite="100001"
        targetSite=""
        time="22:03:34"
        version="Teamcenter P10000.0.0.20120209.00">

  <TransferFormula elemId="id14">
    <SessionOptions elemId="id15">
      <Option elemId="id16"
              name="is_scope_rules_export"
              value="yes"/>
    </SessionOptions>
    <TransferMode elemId="id17"
                  gov_classification=""
                  ip_classification=""
                  license_list=""
                  object_name="PLMXMLAdminDataExport"
                  owning_site="id2"
                  project_list=""/>
    <Reason elemId="id18"></Reason>
  </TransferFormula>
  <TraverseRootRefs>#id1</TraverseRootRefs>
</Header>
</TCXML>
```

Step 12: Create a property set

A property set can be used to export additional properties of an object that are not exported by the PLM XML framework by default. For example, the PLM XML export does not export the **last_mod_date** on an item revision by default, but this can be exported using a property set.

Create a property set as follows to export a BOMLine property that is not exported by default:

Property set clauses do not use the **CLASS** hierarchy for the **CLASS** qualifier. Therefore, to export a property on a given object, the object's class name or type name must match the property set clause's primary object name (second column).



PropertySet

Property Set Name: BOMExportWithFiles

Description:

Scope of Property Set: (● Export) ○ Import

Primary Object C...	Primary Object	Relation Type	Related Property O...	Property Acti...
TYPE	BOMLine	PROPERTY	bl_itemfnd0objectId	DO

The **bl_item_fnd0objectId** is exported as follows:

```
<Occurrence id="id7" instancedRef="#id34" associatedAttachmentRefs=
    "#id9 #id15" occurrenceRefs="id18">
...
<UserData id="id6">
<UserValue value="AuN90mkSAABaaA" title="bl_item_fnd0objectId" />
    </UserData>
...
    </Occurrence>
<Occurrence id="id18" instancedRef="#id31" associatedAttachmentRefs=
    "#id20 #id23" occurrenceRefs="id26" parentRef="#id7">
...
<UserData id="id17">
<UserValue value="Q_M90mkSAABaaA" title="bl_item_fnd0objectId" />
    </UserData>
....
    </Occurrence>
<Occurrence id="id26" instancedRef="#id29" associatedAttachmentRefs=
    "#id28" parentRef="#id18">
...
<UserData id="id25">
<UserValue value="gKO90mkSAABaaA" title="bl_it m_fnd0objectId" />
    </UserData>
...
</Occurrence>
```

3. Working with transfer option sets

What are transfer option sets?

Transfer option sets are created in PLM XML/TC XML Export Import Administration and are used to move data between Teamcenter sites.

Transfer option sets are made up of the following items that configure the import or export operation.

Item	Description
Name	Specifies the name of the transfer option set.
Description	Describes the transfer option set.
Remote site information	Shows whether the transfer option set is for a remote site, thus an import. If so, its remote site ID is included. The transfer mode is defined at the remote site.
Transfer mode	Contains the set of rules that configure export operations if a remote site is not selected. • Closure rules The scope of the data translation. • Filter rules Use conditions that apply a finer level of control over which data gets translated along with the primary objects. • Action rules Sets of methods that can be called before, during, and after the translation. • Property sets A mechanism for PLM XML objects to have arbitrary properties in the form of **UserData** elements.
Options	Displays all unique options in the closure rule conditional clauses for the selected transfer mode.

The Teamcenter administrator can set application extension points and rules along with a business context to restrict the use and administration of transfer option sets to certain roles or groups. The inputs for the application extension point are **SiteID** (string), **IsPush** (logical), and **IsExport** (logical). The output is **OptionSet_Name** (string). The Teamcenter administrator then defines application extension rules to display which option sets are available for a given site ID for an import or export. The business context determines which roles or groups are associated with a particular rule.

Create transfer option sets

1. Choose the **TransferOptionSet** node (located in the lower left pane of the PLM XML/TC XML Export Import Administration window).

   The system displays the **TransferOptionSet** pane.

2. Type a unique name for the transfer option set in the **Name** box. This name should clearly identify the context to the user. For remote site sets, append the site name to the end of the name to maintain uniqueness.

3. Type a description of the transfer option set in the **Description** box.

4. If this transfer option set is for a remote site (in other words, an import), select the **Is the option set for a Remote Site?** check box.

   A remote transfer option set is normally imported from a remote site to ensure that it matches the transfer option set at your site. Although you can create a remote site transfer option set on your site and manually synchronize it to the remote site, Siemens Digital Industries Software recommends that you use the import and export utilities (in Teamcenter, they are **tcxml_import** and **tcxml_export**, respectively) to ensure your remote transfer option set is the same at both sites. Once a transfer option set is imported, you must not change the transfer option set name or site reference. However, you can add or remove options as long as the transfer option set at the remote site is updated to reflect the changes you make.

5. If you selected the **Is the option set for a Remote Site?** check box, select the remote site from the **Remote Site ID** list. If the check box is cleared, select the transfer mode you want to use for the export in the **TransferMode** list. The transfer mode you select must have **TC XML** as its output schema format.

6. If you select a transfer mode, the **Option** list is filled with all unique options in the closure rule conditional clauses associated with the transfer mode. The columns in the list are:

   - **Option**

     Specifies the name of the option in the closure rule conditional clauses.

   - **Display Name**

     Specifies the display name given by the user.

- **Default Value**

  Displays the value of the option. It is either **True**, **False**, or **$opt_rev_select**, which contains a list of **possible revision selector values**.

- **Description**

  Displays the brief description given by the Teamcenter administrator.

- **Group Name**

  Displays the group name given by the Teamcenter administrator to help sort the options in appropriate categories.

- **Read Only**

  Shows if the user can change the option values. If it is cleared, the user is allowed to change values. This is set by the Teamcenter administrator.

If the set is for a remote site, the **Option** list is blank, but you can add options for the set. If you add options, export the set to the remote site when you are finished.

Optionally, the Teamcenter administrator can delete an option by selecting it and clicking the − button to right of the **Option** list or add an option by clicking the + button.

Optionally, the Teamcenter administrator can change the precedence of the options by selecting a row in the table and using the ▲ or ▼ buttons to the right of the **Option** list.

7. Click the **Create** button.

   The system saves the transfer option set to the database and it is displayed in the **TransferOptionSet** node.

Edit transfer option sets

You can edit transfer option sets by adding, removing, or changing the options or transfer mode that define the set.

1. Select the node in the **TransferOptionSet** branch that corresponds to the set you want to edit.

2. Modify the definition by changing options, as described in *Create transfer option sets*, or by changing the transfer mode. You cannot modify the name in the **Option** box if that option came from a closure rule. You can only modify user-defined option names.

3. Click the **Modify** button.

The system saves the changes to the database.

Delete transfer option sets

1. Select the node in the **TransferOptionSet** branch that you want to delete.

2. Click the **Delete** button.

   The system displays the **Delete Confirmation** dialog box.

3. Click the **Yes** button to delete the set, or click the **No** button to cancel.

4. Configuring PLM XML export and import

Time format in PLMXML files

Teamcenter import supports both Greenwich Mean Time (GMT) and legacy time format.

- For time information in the PLMXML file in the legacy time format, all Teamcenter versions will import as a local time. For example:

```
<UserList id="id16" type="list">
    <Item value="2015-03-17T03:50:23"></Item>
    <Item value="2015-03-30T09:23:12"></Item>
    <Item value="2015-03-31T11:17:46"></Item></UserList>
```

- For time information in the PLMXML file in the GMT time format:

  - Prior to Teamcenter 10.1.5 and 11.2.0, the time information must be modified to the legacy time format or the import may fail.

  - Beginning in Teamcenter 10.1.5 and 11.2.0, the time format is identified as Greenwich Mean Time (GMT) or legacy time format and imported accordingly.

  For example:

```
<UserList id="id16" type="list">
    <Item value="2015-03-17T03:50:23Z"></Item>
    <Item value="2015-03-30T09:23:12Z"></Item>
    <Item value="2015-03-31T11:17:46Z"></Item></UserList>
```

An export defaults to the Greenwich Mean Time (GMT) time format in the PLMXML file. To export in the local time format, set the **PLMXML_export_local_date** site preference to **true**. Only **date** type properties defined in the **PropertySet** branch and exported as **UserData** are supported with GMT time format.

Use ApplicationRef for external integrations

Use the **ApplicationRef** element to facilitate updates from external applications to Teamcenter.

PLM XML files generated from Teamcenter typically contain the **ApplicationRef** sub-element, which uniquely identifies the associated object represented in the PLM XML file. For example, the following represents an item:

```
<Product id="id100" name="Test Item" accessRefs="#id8"
subType="Item" productId="item001">
<Description>This is a test.</Description>
<ApplicationRef version="EiAtWgaBV6BXnA" application="Teamcenter"
label="EiAtWgaBV6BXnA"></ApplicationRef></Product>
```

Typical attributes of the **ApplicationRef** element include:

label or version	One of these two attributes or a combination of these two uniquely identify the object. Both can point to the UID (unique identifier) of the item being exported.
application	Name of the application where the XML is generated from.

The **ApplicationRef** element is useful in external or third-party application integrations with Teamcenter using PLM XML. You can provide a name in the **application** attribute and a unique identifier for the object in the **label** attribute. As part of a PLM XML import, Teamcenter creates a **PLMAppUID** object that stores the identifier from the **label** along with the corresponding Teamcenter object's UID. This helps in the re-transfer of the same object from the external application during an update. During the update, Teamcenter uses the **PLMAppUID** object to update the object instead of creating a new object. The same principle is used between Teamcenter-to-Teamcenter transfers using PLM XML.

For each object that is exported from Teamcenter, the contents of the **ApplicationRef** element are as follows. (The **application** attribute is **Teamcenter**.)

Product	**label** = **version** = **Item** UID
Product revision	**label** = **Item** UID, **version** = **Item Revision** UID
Dataset	**label** = **version** = **Dataset** UID
External file	**label** = **version** = **ImanFile** UID
Form	**label** = **version** = **Form** UID
InstanceGraph	**label** = top line's BOM view **bl_bomview** UID]/**InstanceGraph**. This is only for **bomwriter**. No **ApplicationRef** is written in a standard PLM XML export (**Tools**→**Export**→**To PLM XML**).
ProductInstance	**label** = **PSOccurrenceThread** UID, **version** = **PSOccurrence** UID
ProductRevisionView	For leaf node, **version** = **PSOccurrencThread** UID, **label** = **Item** UID/ **PSOccurrenceThread** UID. For a **BomViewRevision**, **version** = **BomViewRevision** UID, **label** = **BomView** UID.
ProductView	**label** = top line's **BomView** UID/**Revision Rule** UID/**Variant Rule** UID/BOM

Occurrence	**label** = **BomView** UID/**PSOccurrenceThread** UID chain all the way to the top. If **APN**/**AbsOcc** is present, it forms the uniqueness, so **label** = **BomView** UID/**apn** or **absocc** UID.

Importing objects with the same ID but different types using PLM XML

When importing PLM XML, if an imported object has the same **item_id** value as an existing object, but a different **type** value, the object will not be imported. For example, consider a target system that has an object with an **item_id** value of **002233** and an object **type** of **Item**. If the imported PLM XML contains an object with the same **item_id** value of **002233** but a **type** value of **CustomItem**, the imported object is treated as different from the object in the target system. However, because the target system already has an object with the same **item_id** value, a new object is not created on the target system.

To import objects such as these, ensure that the imported object has the same **type** as the object on the target system and set the imported object's **subType** value to the different object type.

Configuring PLM XML import and export of alternate identifiers between CAD files and Teamcenter

You can manually configure Teamcenter to support the export and import of alternate identifiers when an object is exported or imported. It is assumed that alternate identifier functionality is enabled, which assumes that identifier and ID context types, naming rules and ID context rules, and identifier display rules are configured. To configure types and rules, see the *BMIDE for Data Model Design*.

The following preferences must be set to enable the import and export of alternate identifiers:

- **IdentifierContextTypeDefault**

  Specifies an existing ID context type to be used as a default ID context when an object is imported without an assigned ID context. This site preferences accepts a single string value.

- **IdentifierTypeDefault**

  Specifies an existing identifier type to be used when an object is imported without an identifier type. This site preference accepts a single string value.

PLM XML/Teamcenter object mapping

The following table lists the mapping between PLM XML objects and Teamcenter objects. You can use this information to read and understand PLM XML files.

In some cases, a Teamcenter object can be represented by more than one PLM XML element, depending on how the object is used. Similarly in some cases, a PLM XML element can be represented by more than one Teamcenter object.

More information about Teamcenter Mechatronics Process Management/PLM XML mapping is available in the *Wiring Harness Design Tools Integration With Teamcenter*.

Teamcenter object	PLM XML element
ActivityCost	ActivityCost
ActivityEntryValue	ActivityEntryValue
Allocation	Allocation
AllocationMap	AllocationGroup
AllocationMapRevision	AllocationGroupRevision
AllowedDeviation	AllowedDeviationRelation
Appearance	Occurrence
AppearanceGroup	OccurrenceGroup
AppearanceRoot	ProductView
AppInterfaceType	plmxml_bus:ApplicationInterface
AppGroupBOPLine	Occurrence
Architecture	Product
ArchitectureRevision	ProductRevision
AsBuiltStructure	RealisedProductUsage
AsBuiltStructure	RealisedProductInstance
AssetGroup	Fleet
AuthorizationRule	plmxml_bus:AuthorisationRule Mapped to PLM XML element in the business schema.
BillRate	BillingRate
BOMLine	Occurrence
BOMView Revision	ProductRevisionView
BOMWindow	InstanceGraph
BOPLine	Occurrence
BoundingMultiBox	Bound
CCObject	CollaborationContext
CfgActivityLine	ConfiguredActivity
CharacteristicDefinition	CharacteristicDefinitionRevision
CharacteristicValue	CharacteristicValue

Teamcenter object	PLM XML element
ClosureRule	plmxml_bus:ClosureRule Mapped to PLM XML element in the business schema.
ConfigurationContext	ConfigurationRule
Connection	FlowConnection
Connection_Terminal	Terminal
ConnectionRevision	FlowConnectionRevision
Dataset	DataSet
Dataset	Representation
DC_CmpTool	CompareTool
DC_DitaValFilter	DITAValueFilter
DC_DitaValFilterTbl	DITAValueFilterRow
DC_EditTool	EditTool
DC_GrphcAttrMap	GraphicAttributeMap
DC_GrphcAttrMapTbl	GraphicAttributeRow
DC_NamspacesTbl	NamespaceRow
DC_Procedure	Procedure
DC_ProcedureRevision	ProcedureRevision
DC_PubType	PublicationTyp
DC_PublishTool	PublishTool
DC_RefTopicType	RefTopicTyp
DC_Schema	CMSchema
DC_SchemaRevision	CMSchemaRevision
DC_StyleSheet	StyleSheet
DC_StyleSheetRevision	StyleSheetRevision
DC_StyleType	StyleTyp
DC_TopicType	TopicTyp
DC_TransfPolTbl	TransformPolicyRow
DC_TransfPolicy	TransformPolicy
DC_ViewTool	ViewTool
DC_XmlSchema	XMLSchema
DC_XmlSchemaRevision	XMLSchemaRevision
DCt_GraphicPriority	GraphicPriorityList
DCt_Language	Language
DCt_TopicTypeGrp	TopicTypGroup

Teamcenter object	PLM XML element
DCt_XmlAttrMap	XMLAttributeMap
DCt_XmlAttrMapTbl	XMLAttributeRow
DeepCopyRule	plmxml_bus:DeepCopyRule Mapped to PLM XML element in the business schema.
DesignRevision	DesignRevision
Design	Design
DesignReq	DesignRequirement
DesignReqRevision	DesignRequirementRevision
DesignRevision	DesignRevision
DeviationDoc	Deviation
DeviationDocRevision	DeviationRevision
Discipline	Discipline
DispApplicabilityData	DispositionApplicabilityRelation
DispositionApplicability	DispositionApplicabilityRelation
DispositionType	Disposition
EngChange	Change
EPMAction	WorkflowAction
EPMBRHandler	WorkflowBusinessRuleHandler
EPMBusinessRule	WorkflowBusinessRule
EPMHandler	WorkflowHandler
EPMJob	WorkflowProcess
EPMSignoffProfile	WorkflowSignoffProfile
EPMTask	Task
EPMTaskTemplate	WorkflowTemplate
Extension	plmxml_bus:Method
FaultCode	FaultCode
Filter	plmxml_bus:FilterRule Mapped to PLM XML element in the business schema.
Fnd0ProxyTask	ProxyTask
Folder	Folder
Form	Form
FullText	DataSet
GDEBOPLine	Occurrence
GDEbvr	ProductRevisionView

Teamcenter object	PLM XML element
GDELine	Occurrence
GDELinkLine	Occurrence
GDEOccurrence	GDEInstance
GDEOccurrenceThread	GDEInstance
GeneralDesignElement	GDE
GeneralDesignElementLink	Link
Group	Organisation
GroupMember	OrganisationMember
HideTypeRule	plmxml_bus:HideTypeRule Mapped to PLM XML element in the business schema.
HRN_Accessory	HarnessProduct
HRN_AssemblyPart	HarnessProduct
HRN_Cavity	ConnectorCavity
HRN_CavityPlug	HarnessProduct
HRN_CavitySeal	HarnessProduct
HRN_ConHousing	HarnessProduct
HRN_CoPackPart	HarnessProduct
HRN_Core	WireCore
HRN_Fixing	HarnessProduct
HRN_GeneralWire	HarnessProduct
HRN_GenTerminal	HarnessProduct
HRN_Harness	HarnessProduct
HRN_Module	HarnessProduct
HRN_Slot	ConnectorCavityGroup
HRN_WireProtect	HarnessProduct
IdContextRule	plmxml_bus:IdContextRule Mapped to PLM XML element in the business schema.
Identifier	Identifier
ImanCompoundPropDef	plmxml_bus:CompoundPropDef Mapped to PLM XML element in the business schema.
ImanFile	ExternalFile
ImanGRM	plmxml_bus:RelationRule Mapped to PLM XML element in the business schema.
ImanItemBOPLine	Occurrence
ImanItemLine	Occurrence

Teamcenter object	PLM XML element
ImanParameters	plmxml_bus:MethodOpts Mapped to PLM XML element in the business schema.
ImanQuery	SavedQueryDef
ImanVolume	Vault
Item	ProcessorProduct
Item	Product
Item	Software
ItemRevision	ProcessorProductRevision
ItemRevision	ProductRevision
ItemRevision	SoftwareRevision
LanguagesTbl	LanguageRow
LogBook	LogBook
LogEntry	LogEntry
LogEntryValues	LogEntryValuesRelation
Lot	LotRevision
Activity	MROActivity
AppearanceGroupLine	ConfiguredOccurrenceGroup
AppearanceLine	Occurrence
AppearancePathNode	InstancePath
AppearancePathRoot	InstancePathRoot
OP	Operation
OPRevision	OperationRevision
Process	Process
ProcessRevision	ProcessRevision
Message	FrameSignal
MessageRevision	FrameSignalRevision
ViewLine	Occurrence
Workarea	WorkArea
WorkareaRevision	WorkAreaRevision
NameField	plmxml_bus:NameField Mapped to PLM XML element in the business schema.
NameRule	plmxml_bus:NameRule Mapped to PLM XML element in the business schema.
Network	FlowConnection

Teamcenter object	PLM XML element
Network_Port	Terminal
NetworkRevision	FlowConnectionRevision
NXAgent	ValidationAgent
NXAgentRevision	ValidationAgentRevision
NXCMValDataRevision	ValidationCheckRevision
NXRDDVValDataRevision	ValidationCheckRevision
NXValData	ValidationCheck
OverrideApproval	OverrideApproval
Paragraph	Requirement
ParagraphRevision	RequirementRevision
Part	Product
Participant	Participant
PartMovement	PartMovement
Person	Person
PhysicalCharacteristics	CharacteristicValueForRealisedProductRelation
PhysicalLocation	MROLocation
PhysicalLocationRevision	MROLocationRevision
PhysicalLocationUsage	MROLocationRelation
PhysicalLogEntries	PhysicalLogEntriesRelation
PhysicalPart	RealisedProduct
PhysicalPartRevision	RealisedProductRevision
PhysicalRealization	RealisedFromRelation
POM_imc	Site
POMObject	plmxml_bus:StorageClassDef Mapped to PLM XML element in the business schema.
Processor	ProcessorProductRevision
Revision Processor	ProcessorProduct
ProjectTeam	Organisation
PropertySet	plmxml_bus:PropertySet Mapped to PLM XML element in the business schema.
PropRule	plmxml_bus:PropertyRule Mapped to PLM XML element in the business schema.
PropertyBMOperation	plmxml_bus:PropertyBusinessOperation Mapped to PLM XML element in the business schema.
PSBOMView	ProductRevisionView

Teamcenter object	PLM XML element
PSBOMViewRevision	ProductRevisionView
PSConnection	Connection
PSConnectionRevision	ConnectionRevision
PSOccurrence	ProductInstance
PSOccurrenceThread	ProductInstance
PSSignal	Signal
PSSignalRevision	SignalRevision
PSSignalValue	SignalValue
PSViewType	View
PublishLink	DefinitionRelation
Range	DailyTime
ReleaseStatus	ReleaseStatus
Requirement	Requirement
RequirementRevision	RequirementRevision
ResourceAssignment	ResourceAssignment
ResourcePool	ResourcePool
RevisionRule	RevisionRule
Role	Role
RouteCurve	BSplineCurve
RouteLocation	Location
RouteLocationRevision	LocationRevision
RouteNode	RouteNode
RoutePath	Route
RouteSegment	RouteSection
SchDeliverable	ScheduleDeliverable
Schedule	Schedule
ScheduleMember	ScheduleMember
ScheduleRevision	ScheduleRevision
ScheduleTask	ScheduleTask
ScheduleTaskRevision	ScheduleTaskRevision
SchedulingFixedCost	FixedCost
SchMgtCostFormInfo	FixedCostInfo
SchTaskDeliverable	ScheduleTaskDeliverable
Signoff	Signoff

Teamcenter object	PLM XML element
Seg0Budget	Budget
Seg0BudgetDefinition	BudgetDefinition
ServiceDiscrepancy	ServiceDiscrepancy
ServiceEvent	TCServiceEvent
ServiceGroup	ServiceGroup
Smr0Warning	Warning
Software	Software
Software Revision	SoftwareRevision
SSP0Frequency	Frequency
SSP0FrequencyRevision	FrequencyRevision
SSP0PartApplicabilityData	PartApplicabilityRelation
SSP0ServicePlan	ServicePlan
SSP0ServicePlanRevision	ServicePlanRevision
SSP0ServiceReq	ServiceRequirement
SSP0ServiceReqRevision	ServiceRequirementRevision
SSP0Skill	Skill
SSP0SkillRevision	SkillRevision
SSP0WorkCard	WorkCard
SSP0WorkCardRevision	WorkCardRevision
StructureContext	StructureContext
TaskDependency	ScheduleTaskDependency
TaskExecutionFormInfo	TaskExecutionInfo
TaskSchedulingFormInfo	TaskSchedulingInfo
TCCalendar	Calendar
TCCalendarEvent	CalendarEvent
TC_Project	Project
TimeSheetEntry	TimeSheetRecord
Tool	plmxml_bus:ApplicationToolDef Mapped to PLM XML element in the business schema.
TraceLink	TraceabilityRelation
TransferMode	plmxml_bus:TransferMode Mapped to PLM XML element in the business schema.
TypeBMOperation	plmxml_bus:TypeBusinessOperation Mapped to PLM XML element in the business schema.

Teamcenter object	PLM XML element
TypeCannedMethod	plmxml_bus:TypeCannedMethod Mapped to PLM XML element in the business schema.
User	User
ValidationReq	ValidationRequirement
ValidationReqRevision	ValidationRequirementRevision
ValidationResult	ValidationResult
ValReqResult	ValidationResult
VariantRule	VariantRule

Configure Teamcenter for exporting NX assemblies to PLM XML

To export Teamcenter data that includes NX assemblies to PLM XML, first perform the following steps.

1. Set the **PIE_IMF_FOR_CAD** preference to a value of **No**. Doing so ensures the CAD files are exported in their native format.

2. If you are using the four-tier rich client, log off of Teamcenter, restart your pool manager, and log back on to Teamcenter.

Performing PLM XML delta imports

Delta import overview

PLM XML delta import is typically used for importing modifications of objects previously imported. Delta information is represented using delta XML, which is based on standard PLM XML schema and PLM XML delta schema.

The root element of any delta XML is a **PLMXMLDelta** element.

PLM XML delta import supports the following operations on objects. Accordingly, for each operation PLM XML delta schema defines different elements.

Operation	Delta schema element
add	Add
modify	Modify
replace	Replace
delete	Delete

Each element has some attributes to indicate its existing context and the target objects on which the operation is performed. The **ApplicationRef** element in PLM XML is widely used in delta XML to identify a certain object. The object can be an existing object in the target system. A delta schema provides the **ExternalReference** element as a parent wrapper for the **ApplicationRef** element to represent an object in the target system, which can be referred by delta operation elements. A user can export the primary objects in PLM XML format to obtain the **ApplicationRef** elements and then construct a delta XML to make changes to the objects.

The COTS **incremental_import** transfer mode can be used for importing a delta XML file. A user can also configure a new transfer mode if needed.

Using the Add element

New objects can be added to the system by using the delta import **Add** element. The new objects must be represented as subelements of the **Add** element. A user can add the objects that are supported by normal PLM XML import under the **Add** element.

Following are examples of how to use the **Add** element:

- Add a new item with **Dataset** attached to its item revision:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
   author="unset" time="12:00:00" date="2013-04-26">
<Add id="id1">
<ProductRevision id="id2" revision="A" masterRef="#id7" subType="ItemRevision"
  name="delta_add_rev">
<AssociatedDataSet role="IMAN_specification" id="id9" dataSetRef="#id8" />
</ProductRevision>
<Product id="id7" subType="Item" name="delta_add_item" productId="001" />
<DataSet id="id8" accessRefs="#id4" name="delta_add_ds" version="1" type="Bitmap" />
</Add>
</PLMXMLDelta>
```

- Add a new BOM line to an existing BOM line by referring to an existing product revision:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id4" type="Occurrence">
<!-- ApplicationRef of the target/parent BOMLine to which the child BOMLine is added
-->
</ExternalReference>
<ExternalReference id="id3" type="ProductRevision">
<!--  ApplicationRef of the ItemRevision referred by the new BOMLine   -->
</ExternalReference>
<Add id="id1">
<Occurrence id="id2" instancedRef="#id3" parentRef="#id4">
<UserData id="id5" type="AttributesInContext">
<UserValue title="OccurrenceName" value="delta_add" />
</UserData>
</Occurrence>
</Add>
</PLMXMLDelta>
```

- Attach a form to an item revision:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id10" type="ProductRevision">
<!-- ApplicationRef of the target/parent ItemReivsion to which the new Form is added
-->
<AssociatedForm role="IMAN_specification" id="id12" formRef="#id11" />
</ExternalReference>
<Add id="id13">
<Form id="id11" subType="CSYS Interface Form" name="my form" subClass="CSYS Interface
Form">
<UserData id="id17" type="FormAttributes">
<UserValue title="Description" value="" />
<UserValue title="Name" value="" />
<UserValue title="Status" value="" />
<UserValue title="Type" value="" />
</UserData>
</Form>
</Add>
</PLMXMLDelta>
```

- Attach a dataset to an existing item revision:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id10" type="ProductRevision">
<!-- ApplicationRef of the target/parent ItemReivsion to which the new Dataset is
added -->
<AssociatedDataSet role="IMAN_specification" id="id12" dataSetRef="#id11" />
</ExternalReference>
<Add id="id13">
<DataSet id="id11" name="my dataset" version="1" type="Bitmap" />
</Add>
</PLMXMLDelta>
```

Using the Modify element

Modify element overview

The **Modify** element includes the following operations and an attribute named **op**. The table shows the valid values for **op**.

Modify operations	op valid values
Add attribute values	**add**
Remove values from an attribute	**remove**
Replace attribute values	**replace**
Delete attribute values	**delete**
Modify the **UserData** element	**modifyUserData**

The **targetRef** attribute of the **Modify** element points to the object to be modified. The object is an existing object represented by the **ExternalReference** element.

The **attributeName** attribute of the **Modify** element is the attribute on the target object whose value is to be modified.

The **values** or **valueRefs** attributes of the **Modify** element is the value to be set on the target attribute. The attribute is defined on the target object in PLM XML schema, not the attribute in Teamcenter.

The **UserData** element under the **Modify** element contains the information to be set on the target object. This can be used for modifying Teamcenter object properties, especially for those not defined as object attributes in PLM XML schema.

Modify element with the add operation

The **add** operation adds new objects as an attribute value to another object. The target object can be an existing one or a new one to be created in the delta import session. Similarly, the attribute value for new objects can refer to existing objects or other newly added objects. For structure modification, the operation can add child BOM lines or add attachment lines.

Following are examples of how to use the **Modify** element with the **add** operation:

- Add a new child BOM line to an existing BOM line by referring to an existing item revision:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<Modify targetRef="#id10" attributeName="occurrenceRefs" op="add" valueRefs="#id3" />
<Add>
<ProductInstance id="id20" sequenceNumber="1020" partRef="#id21" />
<ProductRevisionView id="id21" revisionRef="#id101" />
<Occurrence id="id3" parentRef="#id10" instancedRef="#id101" instanceRefs="#id20" />
</Add>
<ExternalReference id="id10" type="Occurrence">
<!--  ApplicationRef of the BOMLine under which the new BOMLine will be aded.  -->
</ExternalReference>
<ExternalReference id="id101" type="ProductRevision">
<!--  ApplicationRef of the ItemRevision referred by the new BOMLine   -->
</ExternalReference>
</PLMXMLDelta>
```

- Add a dataset attachment line to a BOM line:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<Add>
<DataSet id="id1" name="add_ds" memberRefs="#id7" type="Text" />
<ExternalFile id="id7" locationRef="123.txt" format="txt" />
<AssociatedAttachment id="id5" attachmentRef="#id1" role="IMAN_specification" />
</Add>
<Modify attributeName="associatedAttachmentRefs" targetRef="#id6" op="add"
valueRefs="#id5" />
<ExternalReference id="id6" type="Occurrence">
```

```
<!--  ApplicationRef element of the BOMLine under which the new Dataset will be
attached.  -->
</ExternalReference>
</PLMXMLDelta>
```

- Add objects to a folder:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id10" type="Folder">
<!--  ApplicationRef element of the Folder under which objects will be added.  -->
</ExternalReference>
<ExternalReference id="id11" type="Folder">
<!--  ApplicationRef element of the Folder to be added to the other folder.  -->
</ExternalReference>
<Add>
<ProductRevision id="id20" subType="Item Revision" revision="A" masterRef="#id21" />
<Product id="id21" subType="Item" productId="0011" name="my added part" />
</Add>
<Modify targetRef="#id10" attributeName="folderContentRefs" op="add" valueRefs="#id11
#id21" />
</PLMXMLDelta>
```

Modify element with the remove operation

The **remove** operation removes an attribute value. This can be used to remove an attachment line from a BOM line, remove an **ImanFile** instance from a dataset, or remove objects from folder content. The **remove** operation will remove the reference relation between the objects. To remove a BOM line from its parent BOM line, the delta import **delete** operation can be used.

Note:

To clean up the referenced persistent objects from the database, use the delta **delete** operation.

Following are examples of how to use the **Modify** element with the **remove** operation.

- Remove an attachment line from a BOM line:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id7" type="AssociatedAttachment">
<!--  ApplicationRef element of the Attachment Line to be removed.  -->
</ExternalReference>
<ExternalReference id="id3" type="Occurrence">
<!--  ApplicationRef element of the BOMLine whose attachment line will be removed.
-->
</ExternalReference>
<Modify targetRef="#id3" attributeName="associatedAttachmentRefs" op="remove"
valueRefs="#id7" />
</PLMXMLDelta>
```

- Remove an **ImanFile** instance from a dataset:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<Modify targetRef="#id10" attributeName="memberRefs" valueRefs="#id11" op="remove" />
<ExternalReference id="id10" type="DataSet">
<!--  ApplicationRef element of the Dataset whose ImanFile will be removed.  -->
</ExternalReference>
<ExternalReference id="id11" type="ExternalFile">
<!--  ApplicationRef element of the ImanFile to be removed.  -->
</ExternalReference>
</PLMXMLDelta>
```

- Remove objects from a folder:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id10" type="Folder">
<!--  ApplicationRef element of the Folder under which objects will be removed.  -->
</ExternalReference>
<Modify targetRef="#id10" attributeName="folderContentRefs" op="remove"
  valueRefs="#id11 #id12" />
<ExternalReference id="id11">
<!--  ApplicationRef element of the object to be removed from the Folder.  -->
</ExternalReference>
<ExternalReference id="id12">
<!--  ApplicationRef element of the object to be removed from the Folder.  -->
</ExternalReference>
</PLMXMLDelta>
```

Modify element with the replace operation

The **replace** operation replaces a simple attribute value or object reference type value. A BOM line transformation can also be modified by using the **replace** operation. When replacing an object reference type value, the **Replace** element is used. The **targetRef** value of each **Replace** element constructs the **valueRefs** value of the **Modify** element.

Following are examples of how to use the **Modify** element with the **replace** operation.

- Replace an item name:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<Modify op="replace" attributeName="name" targetRef="#id31" values="my new part
name" />
<ExternalReference type="Product" id="id31">
<!--  ApplicationRef element of the Item whose name will be replaced.  -->
</ExternalReference>
</PLMXMLDelta>
```

- Replace BOM line transformation:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference select="Transform" type="Occurrence" id="id26">
<!--  ApplicationRef element of the BOMLine whose transformation will be replaced.
```

```
-->
</ExternalReference>
<Modify targetRef="#id26" op="replace">2 0 0 0 0 2 0 0 0 0 2 0 0 0 0 1</Modify>
</PLMXMLDelta>
```

- Replace the objects of the folder content.

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="5.1"
  date="2006-12-20" time="15:05:14" author="blum">
<ExternalReference id="id2" type="Folder">
<!--  ApplicationRef element of the Folder whose containing objects will be replaced.
-->
</ExternalReference>
<Modify op="replace" attributeName="folderContentRefs" targetRef="#id2"
  valueRefs="#id31 #id41" />
<Replace id="id3" targetRef="#id31">
<Product id="id32" name="test001" subType="Item" productId="001">
<UserData>
<UserValue title="object_desc" value="delta_replace_test001" />
</UserData>
</Product>
</Replace>
<ExternalReference id="id31" type="Product">
<!--  ApplicationRef element of the Item under the Folder to be replaced.  -->
</ExternalReference>
<Replace id="id4" targetRef="#id41">
<DataSet id="id21" name="test002" version="1" type="Bitmap">
<Description>delta_replace_test002</Description>
</DataSet>
</Replace>
<ExternalReference id="id41" type="Product">
<!--  ApplicationRef element of the Item under the Folder to be replaced.  -->
</ExternalReference>
</PLMXMLDelta>
```

Modify element with the delete operation

The **values** or **valueRefs** attribute are omitted if the **op** value is **delete**. The **target** attribute value is set as empty.

Following is an example of how to modify the **Folder** instance by deleting **Folder** content:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id2" type="Folder">
<!--  ApplicationRef element of the Folder whose content will be deleted.  -->
</ExternalReference>
<Modify id="id1" targetRef="#id2" op="delete" attributeName="folderContentRefs" />
</PLMXMLDelta>
```

Modify element with the ModifyUserData operation

The **attributeName** attribute is omitted if the **op** value is **modifyUserData**. The **UserData** element under the **Modify** element contains the information to be set on the target object. This can be used for modifying Teamcenter object properties, especially for those not defined as object attributes in PLM XML schema.

Following are examples of how to use the **Modify** element with the **modifyUserData** operation.

- Modify user data for a product (**Item** instance):

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id5" type="Product">
<!--  ApplicationRef element of the Item whose properties will be modified.  -->
</ExternalReference>
<Modify targetRef="#id5" op="modifyUserData">
<UserData id="id2">
<UserValue title="object_desc" value="modified" />
</UserData>
</Modify>
</PLMXMLDelta>
```

- Modify user data for an occurrence (**BOMLine** instance):

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id2" type="Occurrence">
<!--  ApplicationRef element of the BOMLine whose properties will be modified.  -->
</ExternalReference>
<Modify id="id1" targetRef="#id2" op="modifyUserData">
<UserData id="id3" type="AttributesInContext">
<UserValue title="SequenceNumber" value="20" />
<UserValue title="OccurrenceName" value="modify_userdata" />
<UserValue title="Quantity" value="2" />
</UserData>
</Modify>
</PLMXMLDelta>
```

- Modify user data for a form:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id5" type="Form">
<!--  ApplicationRef element of the Form whose properties will be modified.  -->
</ExternalReference>
<Modify targetRef="#id5" op="modifyUserData">
<UserData id="id2" type="FormAttributes">
<UserValue title="item_comment" value="modified" />
<UserValue title="user_data_1" value="1" />
<UserValue title="user_data_2" value="2" />
</UserData>
</Modify>
</PLMXMLDelta>
```

Using the Replace element

The delta **Replace** operation replaces an existing object with a new object. The reference relations on the existing object are switched to the new object. The **targetRef** attribute of the **Replace** element is a URI that points to a single **ExternalReference** element, which is used to identify the object to be replaced. The child element of the **Replace** operation is the replacement object. Each **Replace** operation element can process one object only.

The replaced object remains in the database and is not deleted by the delta import.

- Following is an example of how to use the **Replace** element to replace an object with an item:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id1">
<!--  ApplicationRef element of the object to be replaced.  -->
</ExternalReference>
<Replace id="id2" targetRef="#id1">
<Product id="id3" name="test003" subType="Item" productId="003">
<UserData>
<UserValue title="object_desc" value="delta_replace_test" />
</UserData>
</Product>
</Replace>
</PLMXMLDelta>
```

Using the Delete element

The delta **Delete** operation is used to delete specific objects from the database. The **targetRefs** attribute of the **Delete** operation element indicates the URIs of the **ExternalReference** elements to be deleted. The target objects can be a BOM line, a folder, a dataset, a form, an item, or other objects that are supported in normal PLM XML export/import.

Following is an example of how to use the **Delete** element to delete a BOM line:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
  author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference id="id1" type="Occurrence">
<!--  ApplicationRef element of the object to be deleted.  -->
</ExternalReference>
<Delete id="id2" targetRefs="#id1" />
</PLMXMLDelta>
```

Specify a revision rule in delta XML

The revision rule can be applied on the target BOM window during delta import. The syntax uses **XPath** (XML path language) to specify the revision rule on the **ExternalReference** element for an existing BOM line. Only one revision rule can be specified in a delta XML file.

Following is an example of how to delete a BOM line in the context of a given revision rule:

```
<PLMXMLDelta xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
   author="unset" time="12:00:00" date="2013-04-26">
<ExternalReference type="RevisionRule" id="r1">
<!--  ApplicationRef element of the Revision Rule to be applied on the target BOM
Window.  -->
</ExternalReference>
<ExternalReference type="Occurrence" id="d32e7o"
   select="/PLMXMLDelta/ExternalReference[@type='RevisionRule']">
<!-- ApplicationRef element of the BOMLine object to be deleted. -->
</ExternalReference>
<Delete targetRefs="#d32e7o"/>
</PLMXMLDelta>
```

Importing and exporting data in multiple languages

Multi-language support

PLM XML import and export support multiple languages. You can select multiple languages when exporting data using PLM XML, and you can import a PLM XML file that contains more than one language. Non-English locale environment settings are available. Only languages that your system supports are listed. For example, if your system is not Unicode UTF-8 compliant, the list of available languages does not include multi-byte languages.

The language codes in the PLM XML file follow the W3wC standard (for example, **en-us**). It is a five-letter code containing a two-letter language code, a hyphen, and a two-letter area/country code, all in lower case.

The **RFC 4646** and **XML Schema specification** can provide more information.

The language codes within Teamcenter follow the standard Java language naming convention (for example, **en_US** and **fr_FR**). It is a five-letter code containing a two-letter language, an underscore, and a two-letter area code in uppercase. This follows the **ISO639_ISO3166** template.

Sample multilanguage PLM XML file

A multilanguage PLM XML file contains two attributes for specifying the language information: **language** and **languages**. The following shows the portion of a multilanguage PLM XML file that includes these attributes:

```
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema"
  schemaVersion="7.0.0.42"
date="2009-02-20"
time="10:47:51"
author="Teamcenter"
language="en-us"
languages="en-us fr-fr de-de"
```

The **language** attribute specifies the site primary language of the exporting site. the **languages** attribute represents the list of languages (in order) selected by the user during export. In this example,

en-us is the exporting site's primary language, and **en-us**, **fr-fr**, and **de-de** are the languages selected by the user for export. If the PLM XML file does not have a language attribute (or it is unrecognizable), it defaults to **en-us**.

Exporting localized property values

When exporting using the **plmxml_export** and **bomwriter** (for PLM XML format) utilities, use the **-locales** argument to specify the export languages. The site primary language is always included in the export.

Click **Select Languages** to select the languages for the export from the rich client. The **PIE_transfermode_languages** preference specifies the default list of languages that are associated with a transfer mode. These languages are included when the associated transfer mode is used for export, unless otherwise excluded by the user.

If a single-valued property or a multivalued property with an exhaustive LOV (List Of Values) attached is included in a multilanguage export, the display values of the selected LOV value are exported in the requested locale as part of the **Text** element.

Note:

Single-valued properties or multivalued properties with suggestive LOVs are not supported.

If no locales are selected for export, the DB scalar value is exported to PLM XML scalar fields, and no **Text** element is written to the PLM XML file.

When you select multiple languages for export, the database scalar for a given property is exported to the PLM XML scalar field. The language code for this value is specified in the primary attribute of the **Text** element. The **Text** element also contains the translations in various languages as requested by the user. The **Item** element, a subelement of the **Text** element, specifies each language and the corresponding string value. For example, a dataset is created in French and has translations for two other languages (English and German) for its **name** field. The database scalar is French and called **attribute primary**. If you select English and German as the two languages for export, the dataset element in the **.xml** file would look as follows:

```
<DataSet id="id7" name="Porte" nameRef="#id12" accessRefs="#id4"
version="1"
memberRefs="#id9" type="DirectModel">
</DataSet>
<Text id="id12" primary="fr-fr">
     <Item language="en-us">Door</Item>
     <Item language="de-de">Tür</Item>
</Text>
```

In the **.xml** example, the **name** property is exported in two languages, English (**en-us**) and German (**de-de**). Because French is the database scalar (attribute primary), it is exported by default to a PLM XML scalar field. The primary attribute on the **Text** element is marked **fr-fr**, designating that the PLM

XML scalar field contains the French value. The other requested languages, English and German, are also exported in the **Text** element.

Siemens Digital Industries Software recommends that the name of the scope rule (for example, **TransferMode**, **ClosureRule**, **TransferOptionSet**) not be marked as localizable. This may cause issues as some custom application integration or scripts may refer to the scope rules by name. PLM XML/TC XML Export Import Administration shows a **Localization** button if the description of the scope rules are marked as localizable. This works the same way as the **Localization** button in the **Properties** dialog box.

Importing localized property values

You can import a PLM XML file with a site primary language different from that of the importing system. To prohibit this type of import, set the **PIE_allow_import_with_different_sml** preference to false. If the **.xml** file does not contain a language attribute value, or if it is **en** (in the case of third-party **.xml** files), PLM XML treats the language value as **en-us** and proceed with the import.

During import, unsupported languages are ignored and noted in the import log file.

The PLM XML delta schema does not have multilanguage support.

During import, the primary language specified in the PLM XML file is imported as database scalar. This ensures that the scalar value imported at the importing site is the same as the scalar value exported from exporting site. If the primary language is not supported at the importing site, it takes the first supported language from the **Item** list of the **Text** element and treat it as the scalar value (attribute primary).

Adding localized values for option set display name, description, and group name

You can provide localized values for your option set display names, descriptions, and group names by providing the attribute key and its localized value in the **webstrings.xml** properties file for the locale. These properties files are in the Teamcenter Web application file (**tc.war** file by default) at the following location: *teamcenter\dhtml\common\intl\locale\webstrings.xml*

The attribute key must be in the following format:

*Transfer-option-set-name*_*Site-name*_*Option-name*_**displayName**
*Transfer-option-set-name*_*Site-name*_*Option-name*_**description**
*Transfer-option-set-name*_*Site-name*_*Option-name*_**groupName**

For example, to provide localized values for option set attributes in German, explode the **tc.war** file and locate the following file: *teamcenter\dhtml\common\intl\de\webstrings.xml*

To provide localized values for the **op_jt_ds** option of the **TIEOptionSetDefault** option set defined at the **tc820ms1** site, add the following elements to the file:

```
<web_string>
   <name>TIEOptionSetDefault_tc820ms1_opt_jt_ds_groupName</name>
   <string>German-group-name-string</string>
</web_string>
<web_string>
   <name>TIEOptionSetDefault_tc820ms1_opt_jt_ds_displayName</name>
   <string>German-display-name-string</string>
</web_string>
<web_string>
   <name>TIEOptionSetDefault_tc820ms1_opt_jt_ds_description</name>
   <string>German-description-string</string>
</web_string>
```

Archive or regenerate the **tc.war** file and deploy the modified file.

5. Customizing PLM XML import/export extensions

Available customization points

You can use extensions points to customize the import and export of PLM XML files. The following customization points are available:

Business object	Operation name (as listed in the Business Modeler IDE)	ITK PIE registration function	Description
PLM XML	**BMF_PLMXML_register_filter**	**PIE_register_user_filter**	Allows you to register custom filter rules used during import/export. To call this extension point, define a filter rule in PLM XML/TC XML Export Import Administration and use the filter in the **TransferMode** definition. This function is called every time the specific object of interest is encountered.
PLM XML	**BMF_PLMXML_register_actions**	**PIE_register_user_action**	Allows you to register custom action rules used before, during, or after import/export. To call this extension point, define an action rule in PLM XML/TC XML Export Import Administration and use the action in the **TransferMode** definition.
PLM XML	**BMF_PLMXML_register_export_methods**	**PIE_register_user_methods**	Allows you to register custom export methods. This function overrides the default export method and is called if the specific context is used. This function must provide the export code for the mapped object.
PLM XML	**BMF_PLMXML_register_import_methods**	**PIE_register_user_methods**	Allows you to register custom import methods. This function overrides the default import method and is called if the specific context is used. This function must provide the import code.
PLM XML	**BMF_PLMXML_register_schema_mappings**	**PIE_register_user_schema**	Allows you to map Teamcenter and PLM XML objects. It is applicable for exports (import mapping uses PLM XML **subType** attribute).

Following is the order user extensions are called when exporting a PLM XML file:

1. **pre-action**
2. Export process
   - Filters
   - User methods
3. **during-action** (PLM XML file not saved; XML document loaded)
4. **xslt** (optional)
5. **post-action**

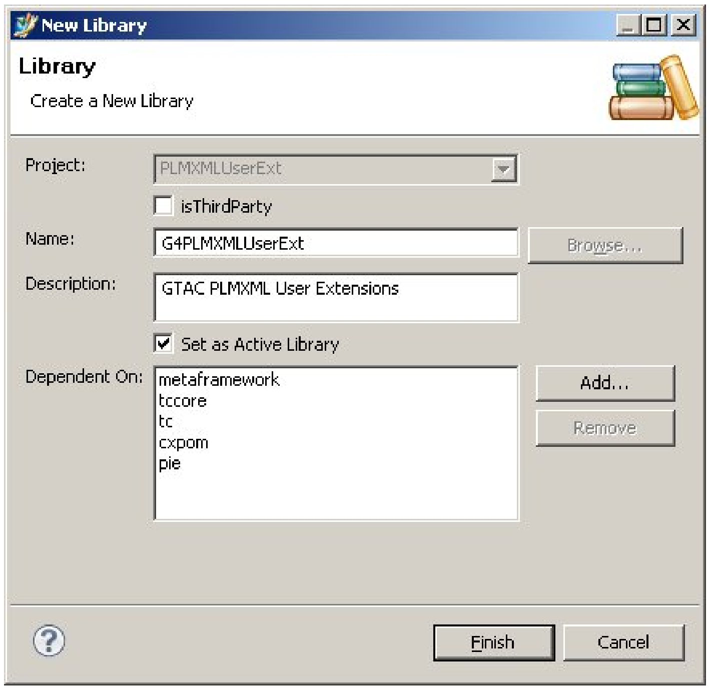
Following is the order user extensions are called when importing a PLM XML file:

1. **pre-action**
2. **xslt** (optional)
3. Load/parse the PLM XML document
4. **during-action**
5. Import process
   - Filters
   - User methods
6. **post-action**

To begin, you must create a Business Modeler IDE project using the foundation template. The project name used in this example is **PLMXMLUserExt**, and it uses a prefix of **G4**.

Set up the Business Modeler IDE PLM XML extension

1. In the Business Modeler IDE, open the **Advanced** perspective by choosing **Window→Open Perspective→Other→Advanced**.
2. Create a new library using the following options.



Library

Create a New Library

Project: PLMXMLUserExt

☐ isThirdParty

Name: G4PLMXMLUserExt

Description: GTAC PLMXML User Extensions

☑ Set as Active Library

Dependent On: metaframework, tccore, tc, cxpom, pie

3. Add the **pie** dependency.

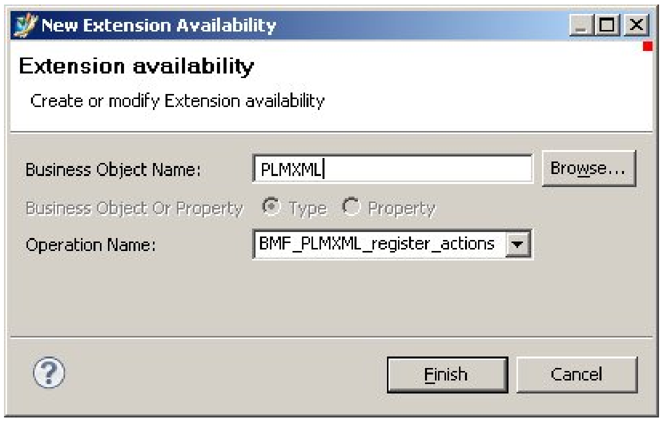
   This adds the PLM XML library.

4. Click **Finish**.

5. In the Business Modeler IDE **Extensions** view, choose **Rules→Extensions**. Right-click **Extensions** and choose **New Extension Definition**.

6. Type **G4PLMXMLUserExt** as the extension name and click **Add**.

7. Complete the **New Extension Availability** dialog box.



Extension availability

Create or modify Extension availability

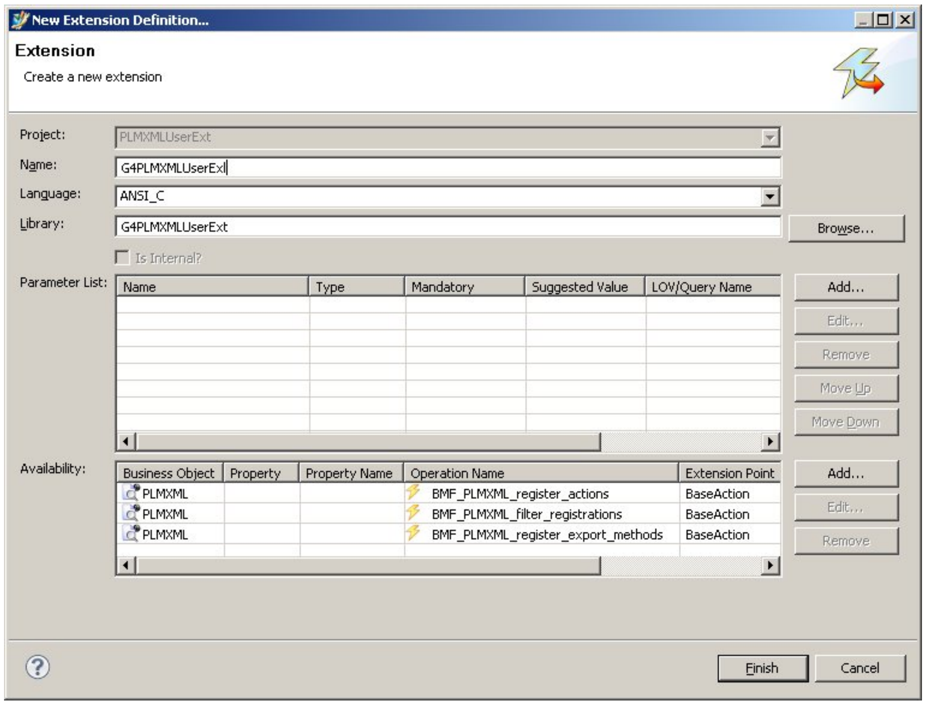
Business Object Name: PLMXML

Business Object Or Property: [✓ Type] Property

Operation Name: BMF_PLMXML_register_actions

8. Add a new extension availability for filters and export methods.

The **New Extension Definition** dialog box should look as follows:



Project: PLMXMLUserExt
Name: G4PLMXMLUserExt
Language: ANSI_C
Library: G4PLMXMLUserExt
Is Internal?

Parameter List:

Name	Type	Mandatory	Suggested Value	LOV/Query Name

Availability:

Business Object	Property	Property Name	Operation Name	Extension Point
PLMXML			BMF_PLMXML_register_actions	BaseAction
PLMXML			BMF_PLMXML_filter_registrations	BaseAction
PLMXML			BMF_PLMXML_register_export_methods	BaseAction

9. Click **Finish**.

   The **Extensions** view shows the new extension.

10. In the **Extensions** view, expand **User Exits** and double-click **PLMXML**.

11. Expand the **Operations** folder and select **BMF_PLMXML_register_actions**. Add **G4PLMXMLUserExt** to the **Base-Action** list.

12. Add **G4PLMXMLUserExt** to the **Base-Action** list for the **BMF_PLMXML_filter_registrations** and **BMF_PLMXML_register_export_methods** operations.

13. Save the data model by choosing **File→Save Data Model**.

Generate the Business Modeler IDE PLMXML extension code

1. In the Business Modeler IDE, open the **Advanced** perspective by choosing **Window→Open Perspective→Other→Advanced**.

2. In the **Extensions** view, right-click **G4PLMXMLUserExt** and choose **Generate extension code**.

3. In the **Navigator** View, select the root project node and press F5 to refresh the project.

4. Expand the **output→server→gensrc→G4PLMXMLUserExt** folder and open the **G4PLMXMLUserExt.c** file.

   The **.c** file includes a function similar to the following:

```
int G4PLMXMLUserExt(METHOD_message_t *msg, va_list args)
{
                 return 0;
}
```

5. To call the PLM XML registration code, update the **G4PLMXMLUserExt.c** file as follows:

```
#include <pie/pie.h>
#include <tccore/tctype.h>
#include G4PLMXMLUserExt/G4PLMXMLUserExt.h>
#define CUSTOM_PIE_CONTEXT_STRING   "CUSTOM_PIE_CONTEXT_STRING"
int G4PLMXMLUserExt(METHOD_message_t *msg, va_list args ) //ignore
argument list.
{
   PIEUserMethodList_t userMethodList[] = {
   {CUSTOM_PIE_CONTEXT_STRING, "ProductRevision",
   (PIE_user_method_func_t)G4UserImportExtension}, // PLMXML
classname.
   {CUSTOM_PIE_CONTEXT_STRING, "ItemRevision",
   (PIE_user_method_func_t)G4UserExportExtension} }; // Teamcenter
classname.

   printf("G4PLMXMLUserExt registration\n");
   /* user actions */
   PIE_register_user_action("GTACCustomExportPreAction",
   (PIE_user_action_func_t)G4PreExportExtension);
   PIE_register_user_action("GTACCustomExportDuringAction",
   (PIE_user_action_func_t)G4DuringExportExtension);
   PIE_register_user_action("GTACCustomExportPostAction",
   (PIE_user_action_func_t)G4PostExportExtension);
   /* filters */
   PIE_register_user_filter("GTACCustomFilterRule",
   (PIE_user_filter_func_t)G4FilterExtension);
   /* (override) import/export methods */
   PIE_register_user_methods(userMethodList, 2);
   return 0;
}
PIE_rule_type_t G4FilterExtension(void* userPath)
{
   tag_t tag = NULLTAG;
   tag_t type_tag = NULLTAG;
   char type_name[TCTYPE_name_size_c + 1] = { 0 };
   printf("G4FilterExtension\n");
   PIE_get_obj_from_callpath(userPath, &tag);
   TCTYPE_ask_object_type(tag, &type_tag);
   TCTYPE_ask_name(type_tag, type_name);
   printf("Filter Object Type: %s\n", type_name);
   return PIE_process; // or skip
}
int G4PreExportExtension(tag_t session)
{
          printf("G4PreExportExtension\n");
          // .xslt file used to beautify PLMXML file for easy
reading.
          PIE_session_set_user_xslt_file(session,
          "C:\\TCU83\\dev\\xslt\\prettyprint.xsl");
          return 0;
}
int G4DuringExportExtension(tag_t session)
{
          printf("G4DuringExportExtension\n");
          return 0;
}
int G4PostExportExtension(tag_t session)
{
          printf("G4PostExportExtension\n");
          return 0;
}
int G4UserImportExtension(void* userPath)
{
          printf("G4UserImportExtension\n");
          return 0;
}
int G4UserExportExtension(void* userPath)
{
          printf("G4UserExportExtension\n");
          return 0;
}
```

6. Update the **G4PLMXMLUserExt.h** as follows:

```
extern G4PLMXMLUSEREXT_API PIE_rule_type_t
                                         G4FilterExtension(void*
userPath);

extern G4PLMXMLUSEREXT_API int G4PreExportExtension(tag_t session);
extern G4PLMXMLUSEREXT_API int G4DuringExportExtension(tag_t
session);
extern G4PLMXMLUSEREXT_API int G4PostExportExtension(tag_t session);
extern G4PLMXMLUSEREXT_API int G4UserImportExtension(void* userPath);
extern G4PLMXMLUSEREXT_API int G4UserExportExtension(void* userPath);
extern G4PLMXMLUSEREXT_API int G4PLMXMLUserExt
                                    (METHOD_message_t* msg, va_list
args);
```

7. Save the source files and choose **Project→Build All**.

   Your project should build without errors.

Arguments for customizing PIE functions

Customizing PIE functions overview

You can use arguments to customize PIE functions.

For a full listing of available PIE functions, see the *Integration Toolkit Function Reference*.

PIE_register_user_methods

The **PIEUserMethodList_t** structure is defined in the **pie/pie.h** header and is defined as follows:

```
typedef struct PIEUserMethodList_s
{
     const char                *contextStr;    /* Context String */
     const char                *name;          /* Class or type name */
     PIE_user_method_func_t    func;           /* User method */
} PIEUserMethodList_t;
```

contextStr

Specifies a unique string entered in the **Context** field in the **TransferMode** definition. This string allows the user to group user methods with different names for a given context.

name

Maps an import/export method to a specific Teamcenter object or PLM XML element. For example:

Names used in export user-methods	**Names used in import user-methods**
Teamcenter objects	**PLM XML element**
Item	**ProcessorProduct**
Item	**Product**
Item	**Software**
ItemRevision	**ProcessorProductRevision**

Names used in export user-methods **Teamcenter objects**	**Names used in import user-methods** **PLM XML element**
ItemRevision	**ProductRevision**
ItemRevision	**SoftwareRevision**

For a full list of schema mappings, see *PLM XML/Teamcenter object mapping*.

func

Specifies a function pointer that has a function signature of:

```
int foo_method(void* userPath)
```

The **userPath** argument can be used in any of the **PIE_*** ITK functions that take a void* named **userPath**. The function body generally uses the PLM XML SDK to implement the writing or reading of the PLM XML document.

For a full listing of available PIE functions, see the *Integration Toolkit Function Reference*.

PIE_register_user_filter

filterRuleName

Specifies a unique string that is displayed in the **FilterRule, Filter Rule Name** list.

user_m

Specifies a function pointer with a function signature of:

```
PIE_rule_type_t foo_filter(void* userPath)
```

The **userPath** argument can be used in any of the **PIE_*** ITK functions that takes a void* named **userPath**. Valid return values are defined in the **PIE_rule_type_e enum** (**pie/pie.h**) as follows:

```
typedef enum PIE_rule_type_e
{
     PIE_skip                 = 1,
     PIE_process              = 2,
     PIE_travers_no_process   = 4,
     PIE_travers_and_process = 6
} PIE_rule_type_t;
```

PIE_register_user_action

handleName

Specifies a unique string that is displayed in the **ActionRule, Action handler** list.

user_m

Specifies a function pointer with a function signature of:

```
int foo_action(tag_t session)
```

The session **tag_t** can be used in any of the **PIE_*** ITK functions that take a **tag_t** session.

Deploy the Business Modeler IDE Template

1. In the Business Modeler IDE, open the **Advanced** perspective by choosing **Window→Open Perspective→Other→Advanced**.
2. In the **Extensions** view, right-click the root node and choose **Deploy Template**.
3. Enter your user ID and password and click **Connect**.
4. When the connection is established, click **Finish**.

   When the deployment is complete, a confirmation dialog box appears.

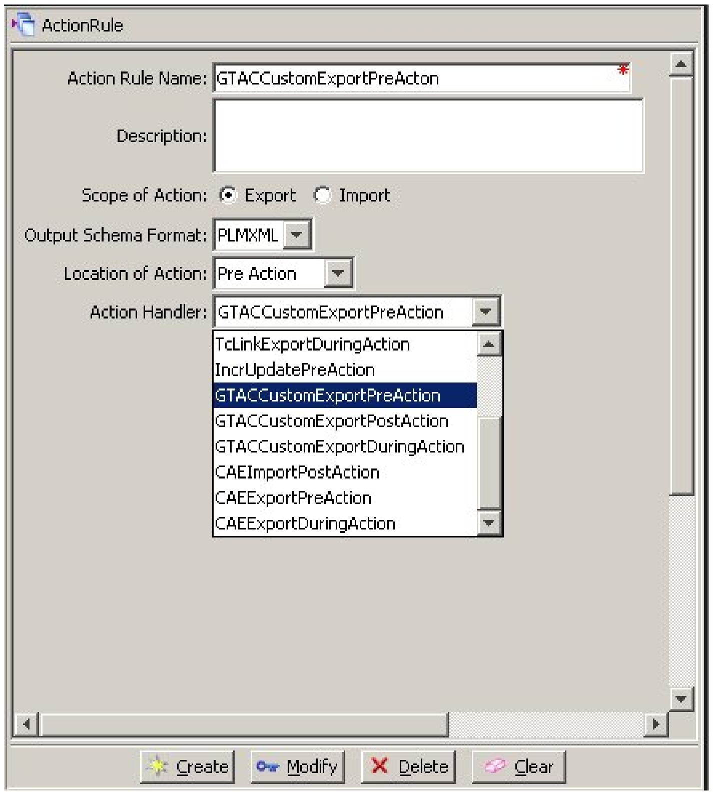
Load the custom extension DLL

To load the custom DLL, you must set the **BMF_CUSTOM_IMPLEMENTOR_PATH** preference.

1. Choose **Edit→Options**.
2. Enter the path in the **Current Values** box and click **OK**.
3. Restart Teamcenter.

Create a custom transfer mode

1. Open the PLM XML/TC XML Export Import Administration application.
2. Click **ActionRule** in the lower left pane.
3. Create a pre-action rule by completing the **ActionRule** pane.



Action Rule Name: GTACCustomExportPreActon

Description:

Scope of Action: [✓ Export] Import

Output Schema Format: PLMXML

Location of Action: Pre Action

Action Handler: GTACCustomExportPreAction

- TcLinkExportDuringAction
- IncrUpdatePreAction
- GTACCustomExportPreAction
- GTACCustomExportPostAction
- GTACCustomExportDuringAction
- CAEImportPostAction
- CAEExportPreAction
- CAEExportDuringAction

4. Click **Create**.

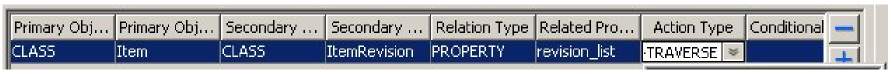
5. Create export actions for **During** and **Post-action** extensions.

6. Click **ClosureRule** in the lower left pane.

7. In the **Traversal Rule Name** box, enter **GTACCustomClosureRule**.

8. Set **Scope of Traversal** to **Export** and select **PLM XML** from the **Output Format** list.

9. In the table, click the **+** button to create a new row and enter the following.



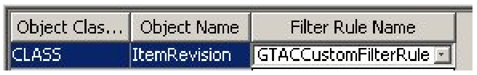
Primary Obj...	Primary Obj...	Secondary ...	Secondary ...	Relation Type	Related Pro...	Action Type	Conditional
CLASS	Item	CLASS	ItemRevision	PROPERTY	revision_list	TRAVERSE	

10. Click **Create**.

11. Click **FilterRule** in the lower left pane.

12. In the **Filter Rule Name** box, enter **GTACItemRevExportFilter**. Select **Export** as the **Scope of Filter** and select **PLM XML** from the **Output Schema Format**.

13. In the table, click the **+** button to create a new row and enter the following.

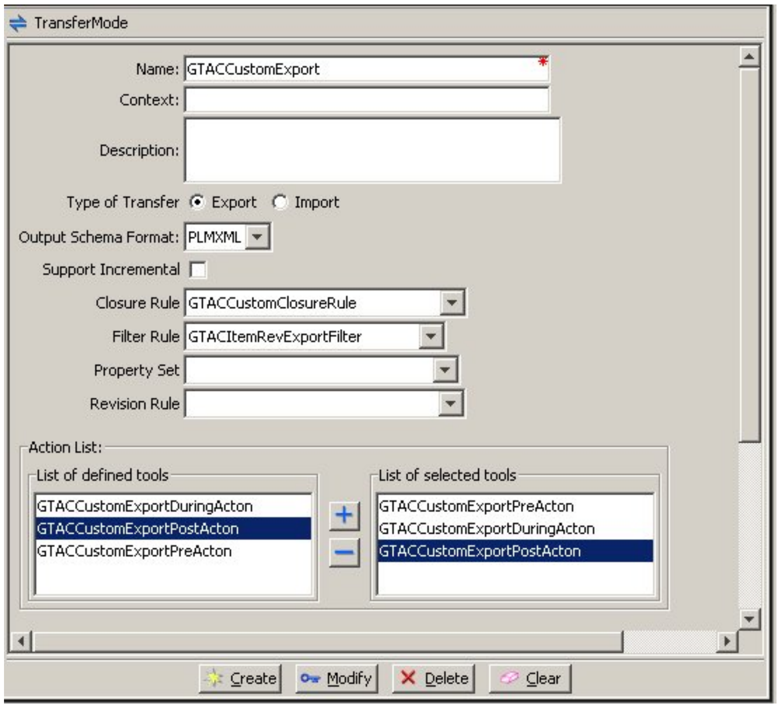


Object Clas...	Object Name	Filter Rule Name
CLASS	ItemRevision	GTACCustomFilterRule

14. Click **Create**.

15. Click **TransferMode** in the lower left pane.

16. Complete the **TransferMode** pane.



TransferMode

Name: GTACCustomExport
Context:
Description:
Type of Transfer [✓ Export] Import
Output Schema Format: PLMXML
Support Incremental
Closure Rule GTACCustomClosureRule
Filter Rule GTACItemRevExportFilter
Property Set
Revision Rule

Action List:

List of defined tools:
GTACCustomExportDuringActon
GTACCustomExportPostActon
GTACCustomExportPreActon

List of selected tools:
GTACCustomExportPreActon
GTACCustomExportDuringActon
GTACCustomExportPostActon

17. Click **Create**.

Export a custom PLM XML file using a custom transfer mode

1. In My Teamcenter, create a new item called **Export**.

2. Choose **Tools→Export→To PLMXML**.

3. Click **OK**.

   The **.xml** file of the item and item revision export should look similar to following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- GENERATED BY: PLM XML SDK 7.0.2.173 -->
<PLMXML xmlns="http://www.plmxml.org/Schemas/PLMXMLSchema" schemaVersion="6"
language="en-us" date="2011-06-03" time="10:45:32"
author="Teamcenter V8000.3.0.20100916.00 - username@IMC--2118502821(-2118502821)">
   <Header id="id1" traverseRootRefs="#id2" transferContext="CustomExport"/>
```

```
      <ProductRevision id="id6" name="Export" accessRefs="#id3" subType="ItemRevision"
      masterRef="#id2" revision="A">
        <ApplicationRef version="wJC1OqqpYs6Y1C" application="Teamcenter"
         label="wFJ1OqqpYs6Y1C"/>
      </ProductRevision>
      <Product id="id2" name="Export" accessRefs="#id3" subType="Item"
      productId="000025">
        <ApplicationRef version="wFJ1OqqpYs6Y1C" application="Teamcenter"
        label="wFJ1OqqpYs6Y1C"/>
      </Product>
      <AccessIntent id="id3" intent="reference" ownerRefs="#id4"/>
      <Site id="id4" name="IMC--2118502821" siteId="-2118502821">
        <ApplicationRef version="wxPxfJW5Ys6Y1C" application="Teamcenter"
        label="wxPxfJW5Ys6Y1C"/>
        <UserData id="id5">
           <UserValue value="1" title="dbms"/>
        </UserData>
      </Site>
   </PLMXML>
```

The default schema mapping is **ProductRevision-to-ItemRevision** and **Product-to-Item**.

Creating custom transformation rules

Custom transformation rules

You can create custom transformation rules using *.csv* rules files that map XML structures, elements, and attributes from one XML format to another. The transformation rules can then be attached to a specific transfer mode using the **plmxml_tm_edit_xsl** utility. A typical use is sharing information between Teamcenter and other systems by transforming between TC XML and either PLM XML or AP 242 XML.

Refer to *Exporting and importing using PLM XML* for details on working with PLM XML/TC XML Export Import Administration.

Transformation rules are of the following types:

- Mapping rules that relate source structures to target structures
- Traverse rules that describe a traversal from one node to another node, or from one attribute to another attribute.
- Function rules that perform the actual transformation to values or objects.

Create mapping rules directly in a *.csv* file as described in **Create custom transformation rules directly in a .csv file**.

Use the Diverse Schema Utility to create mapping rules when Briefcase is configured to use low-level TC XML to exchange data between Teamcenter sites as described in **Using Advanced Multi-Schema Exchanger to create mapping rules**.

Create custom transformation rules directly in a .csv file

Complex mapping rules can require transformations beyond mapping item types and attributes with the Diverse Schema Utility. In these situations, you can create the mapping rules directly in a *.csv* file using the following process.

1. Map the transformations of the structures, elements, and attributes in the original XML source files to their related structures, elements, and attributes in the target XML files.

   - Refer to *Custom transformation basic syntax* for syntax rules for common transformation tasks.

   - Refer to *Custom transformation rule advanced syntax* for syntax rules for less common transformation tasks.

   - Refer to *XML transformation tool rule examples* for transformation rule examples.

   Create the files with an ASCII editor or with a spreadsheet, saving the files as comma-delimited (*.csv*) ASCII files.

   Place any *.csv* files containing includes in the *<tc_data>\xformer* directory.

2. Attach the transformation rules to a transfer mode using the **plmxml_tm_edit_xsl** utility.

   For details on working with transfer modes, see *Using transfer modes* in the *Classification Admin* guide.

   For details on working with the **plmxml_tm_edit_xsl** utility, see *plmxml_tm_edit_xsl* in the *Utilities Reference* guide.

3. Use Teamcenter to export sample data to PLM XML to test your transformation rules. Adjust your transformation rules as necessary and retest.

Creating mapping rules using the Advanced Multi-Schema Exchanger

Using Advanced Multi-Schema Exchanger to create mapping rules

Use Advanced Multi-Schema Exchanger to create mapping rules to transform data when it is transferred between Teamcenter sites using different schemas. You can create mapping rules for the item types in your source schema or for subsets of item types in the schema as defined by schema subsets attached to Briefcase files.

Once you have created item type mappings, you can validate the mapping rules using Advanced Multi-Schema Exchanger. Validation allows you to identify potential issues before actually importing the data. After you have addressed the issues, use Advanced Multi-Schema Exchanger to attach the mapping rules to the transfer modes used to transfer the data between sites.

Preparing for a mapping project

The following items are required for creating mapping rules using Advanced Multi-Schema Exchanger:

Source schema or schema subset

The source schema describing the data being exchanged. This can be **a full JSON schema** covering all possible item and attributes, or one or more Briefcase files containing **a subset of the full schema** covering only the items to be exchanged.

Target schema

The target JSON schema describing the data in the target database.

Data example files

(Optional) One or more Briefcase files including examples of items representing all schema differences for which mapping is required.

Creating a JSON schema file

The **tc_janus_schema** command line utility queries the Teamcenter database and creates a JSON schema file for the class, types, item types, and administrative data in the database. Use JSON schema files from the source and target sites to **map gaps (differences) between the source and target schemas**. If you have security or project scoping concerns, you can also **share only a portion of the source schema with the target site**.

The **tc_janus_schema** utility runs on Windows and Linux platforms. GNU libc 2.14 or higher is required when running the utility on Linux.

Create a JSON schema file from your Teamcenter database as follows:

1. From a Teamcenter command window, navigate to *TC_ROOT\janus\tc_schema*.

2. Run the **tc_janus_schema** utility using the following form:

   ```
   tc_janus_schema Tc-admin-user password group output_schema_file.json
   ```

   Tc-admin-user, *password*, *group*

   Credentials of a Teamcenter administrator who is a member of a group with DBA privileges

   output_schema_file

   The name of the resulting JSON schema file

Working with data models using non-UTF-8 character sets

If you receive "codec can't decode byte" errors when running the **tc_janus_schema** utility, verify the character set used by your data model.

The **tc_janus_schema** utility leverages the character set supported by the Teamcenter server and database. By default, the character set is UTF-8. Confirm the character set used on your Teamcenter server and update your **tc_janus_schema** utility configuration to handle non-UTF-8 character sets.

1. From your a Teamcenter command window for the Teamcenter environment, run the following command:

   ```
   find_system_supported_encoding -outputFile=tc.txt
   ```

2. Open *tc.txt* in an ASCII editor. If the language stated in the file is "UTF-8", no further steps are necessary. Otherwise, note the character set value and continue with this procedure.

3. In an ASCII editor, open the file *TC_ROOT\janus\tc_schema\config.json*.

4. In the following list, locate the character set value reported in *tc.txt*. Update the value of **tc_char_set** in *config.json* to the **Codec** value for that character set. For example, if your Teamcenter character set is "windows-1252", set **tc_chart_set** "cp1252".

Character Set	Codec	Languages
big5-tw, csbig5	**big5**	Traditional Chinese
932, ms932, mskanji, ms-kanji	**cp932**	Japanese
949, ms949, uhc	**cp949**	Korean
950, ms950	**cp950**	Traditional Chinese
windows-1252	**cp1252**	Western Europe
eucjp, ujis, u-jis	**euc_jp**	Japanese
euckr, korean, ksc5601, ks_c-5601, ks_c-5601-1987, ksx1001, ks_x-1001	**euc_kr**	Korean
chinese, csiso58gb231280, euc-cn, euccn, eucgb2312-cn, gb2312-1980, gb2312-80, iso-ir-58	**gb2312**	Simplified Chinese
936, cp936, ms936	**gbk**	Unified Chinese
gb18030-2000	**gb18030**	Unified Chinese
iso-8859-2, latin2, L2	**iso8859_2**	Central and Eastern Europe
iso-8859-5, cyrillic	**iso8859_5**	Bulgarian, Belorussian, Macedonian, Russian, Serbian
iso-8859-8, hebrew	**iso8859_8**	Hebrew
iso-8859-15, latin9, L9	**iso8859_15**	Western Europe
csshiftjis, shiftjis, sjis, s_jis	**shift_jis**	Japanese

If "codec can't decode byte" errors continue to be reported when running **tc_janus_schema** after updating to the correct codec value, change the value of **decode_errors** in *config.json* to **ignore**.

Creating a schema subset

For security or project scoping reasons, you may wish to share only a portion of the source schema with the target site. This situation is handled using the **briefcase_add_mapping_info** command line utility to attach the pertinent portions of the full JSON schema file to one or more Briefcase files. Use the resulting Briefcase files to map gaps between the schema subset and the target schema.

The **briefcase_add_mapping_info** utility analyzes and provides schema definitions for the class, types, item types, and administrative data in Briefcase (*.bcz*) files. Schema information for data not included in the *.bcz* files is excluded.

You can run the **briefcase_add_mapping_info** utility on Windows and Linux platforms. GNU libc 2.14 or higher is required when running the utility on Linux.

Use the following steps to share a subset of the source schema.

1. Create one or more *.bcz* files that contain the set of objects that represent the schema subset to be shared. The files should contain examples from all item types that may be exchanged with the target site so that mapping rules can be created for all gaps between the schemas.

   Place these files in a single directory.

2. Open a Teamcenter command window from either the Teamcenter environment in which the *.bcz* files were created or from another Teamcenter environment with the same data models.

   Navigate to the following directory: *TC_ROOT\janus\tc_schema*.

3. Run **briefcase_add_mapping_info** using the following form:

   ```
   briefcase_add_mapping_info Tc-admin-user password -g=group
   folder_name_or_bcz_file_names json_schema_file
   ```

   Where:

   Tc-admin-user, *password*, *group*

   Credentials of a Teamcenter administrator who is a member of a group with DBA privileges.

   folder_name_or_bcz_file_names

   The path to the folder containing the low-level TC XML *.bcz* files to be analyzed.

   Alternately, you can specify a list of one or more comma-separated Briefcase files. Specify either a folder name or individual file names, but not both.

   json_schema_file

(Optional) The full JSON source schema file. If the schema file is not specified, **briefcase_add_mapping_info** uses data directly from Teamcenter to create the schema subset.

The **briefcase_add_mapping_info** utility analyzes the Briefcase files. For each Briefcase file, the utility creates a new file with *_mapping* appended to the base name. This file contains the contents of the original file with pertinent parts of the schema attached. For example, running the utility on *BCZ_000232__IMC-10002.bcz* produces the file *BCZ_000232__IMC-10002_mapping.bcz*.

Be aware of the following items when running the utility:

- Subfolders are ignored.
- Only low-level TC XML Briefcase files are processed. Others are skipped.
- Files with attached schema information are skipped.
- If a *_mapping.bcz* file already exists in the directory, the Briefcase file with the same base name is skipped. Delete existing files with schemas attached when reprocessing Briefcase files.

If you receive "codec can't decode byte" errors when running **tc_janus_schema**, verify the character set used using the steps in *Working with data models using non-UTF-8 character sets* in **Creating a JSON schema file**.

Creating a mapping project

A collection of Advanced Multi-Schema Exchanger schema mapping rules is called a project. Create a new mapping project for defining the set of mapping rules required for exchanging data between two sites.

Use the following procedures to configure your project.

Create a project folder

The files supporting a project are stored in a project folder you create. To begin a new project, create a new empty folder. If you have an existing project folder, you can copy it and use its rules as the basis for a new project.

Start the Advanced Multi-Schema Exchanger

Open a Teamcenter command prompt and navigate to the following subdirectory of your Teamcenter installation directory (*TC_ROOT*):

```
TC_ROOT\janus
```

From that directory, run *janus.exe* to open the Advanced Multi-Schema Exchanger.

Specify your project properties

1. In the **Overview** tab, under **Properties**, click **Edit** to enable editing of the project properties.

2. Set **Project Directory** to the directory you created for the project.

3. If you have a source JSON schema for the objects you are mapping, set **Source Schema** to that file.

   Alternatively, you can map objects based on a subset of the source schema attached to one or more Briefcase files. See *Add the Briefcase files* later in this section.

4. Set **Target Schema** to the target JSON schema to which you are mapping objects.

5. Under **Properties**, click **Stop Edits** to save the project settings.

   Teamcenter Properties (environment and credentials) do not need to be specified for mapping, but must be specified for validating and deploying mapping rules.

Add the Briefcase files

On the **Mapping** tab, click **Add File** to add all of the Briefcase or TC XML files containing data for which mapping is necessary. The files are added to the **Source Files** list.

If you did not specify a source schema, a source schema based on the objects in the Briefcase files is created. See Creating a schema subset for more information on schemas attached to Briefcase files.

Rule mapping process

Resolve differences between the source schema and the target schema by mapping the **Source Files** items, datasets, forms, and other objects between the schemas.

When you map an object in the source schema to an object in the target schema (using the **Forward** tab), the Advanced Multi-Schema Exchanger automatically creates a mapping from the target object to the source object (on the **Reverse** tab) for use when re-importing data. If a source object had been mapped to multiple target objects on the **Forward** tab, those relationships are maintained when mapping target objects on the **Reverse** tab.

As you map objects, **Status** values increase. When you have created mapping rules for all objects, all Briefcase file **Status** levels show a value of 100%. (Status values are not reported for general mapping tasks.) You can then validate the mapping rules, deploy the mapping rules by attaching them to a transfer mode, and export the data.

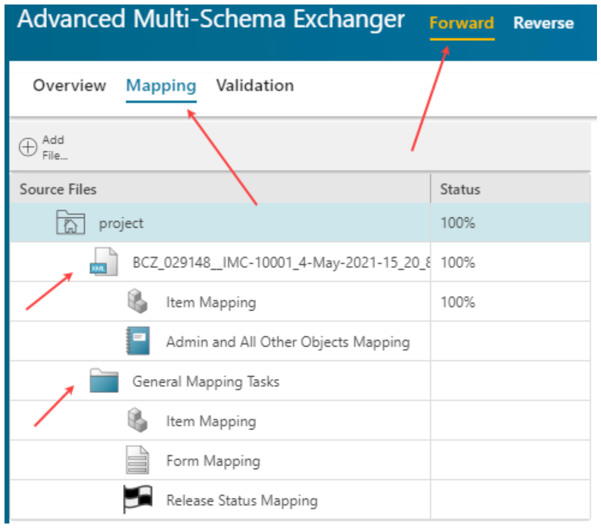
Map objects of the following types:

Mapping Type	Description
Item	Standard Teamcenter items
Admin and All Other Objects Mapping	Administrative objects such as **ClosureRule**, **Group**, **PropertySet**, and other objects of the closure rules
Form	Forms and their attributes
Release Status	Release status objects
GRM	**ImanRelation** GRM (generic relationship management) objects
Table Property	Object properties that are table properties

Map these types as described in Mapping object types.

Mapping object types

With your project open in the Advanced Multi-Schema Exchanger, click the **Mapping** tab to show the files and categories containing objects that can be mapped. For example:



Source Files	Status
project	100%
BCZ_029148__IMC-10001_4-May-2021-15_20_8	100%
Item Mapping	100%
Admin and All Other Objects Mapping	
General Mapping Tasks	
Item Mapping	
Form Mapping	
Release Status Mapping	

Each of the Briefcase and TC XML files (*.bcz* files) you have added to your project is listed, as well as a collection of general mapping tasks.

The **General Mapping Tasks** category covers general types that are not necessarily part of an exported Briefcase. If you have no Briefcase files in your project, use the **General Mapping Tasks** types to create your mapping rules.

Use the following guidelines to create mapping rules.

Object mapping

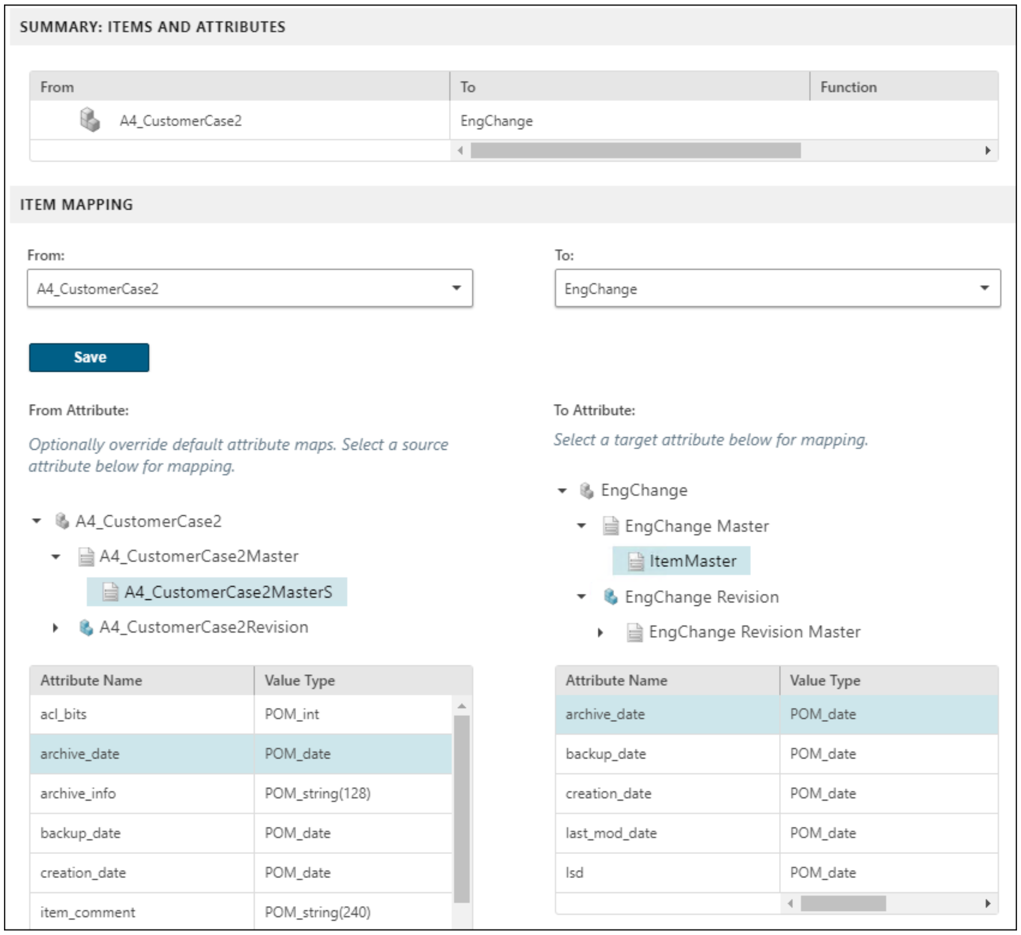
1. Click the **Mapping** tab to show the files and categories containing objects that can be mapped. Ensure the **Forward** tab is selected at the top of the Advanced Multi-Schema Exchanger.

2. To create mapping rules between objects in the source and target schemas, click an object type under a Briefcase file or under **General Mapping Tasks**. A summary of the currently mapped objects and their attributes is displayed on the right side of the Advanced Multi-Schema Exchanger.

3. Edit mapping rules by selecting the object in the **From** list. You can also select objects with existing mapping rules directly from the summary list.

4. In the **To** list, select the target object. Only valid target objects are listed.

   If this is a new mapping, click **Save**. The object's attribute or table properties mappings are displayed, allowing you to override the default mapping rules. For example:



SUMMARY: ITEMS AND ATTRIBUTES

From	To	Function
A4_CustomerCase2	EngChange	

ITEM MAPPING

From: A4_CustomerCase2

To: EngChange

From Attribute:

Optionally override default attribute maps. Select a source attribute below for mapping.

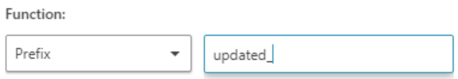
Attribute Name	Value Type
acl_bits	POM_int
archive_date	POM_date
archive_info	POM_string(128)
backup_date	POM_date
creation_date	POM_date
item_comment	POM_string(240)

To Attribute:

Select a target attribute below for mapping.

Attribute Name	Value Type
archive_date	POM_date
backup_date	POM_date
creation_date	POM_date
last_mod_date	POM_date
lsd	POM_date

   To avoid item ID conflicts, consider adding prefixes and suffixes to the target items or forms as part of the mapping. Select a source element, set **Function** to "Prefix" or "Suffix", and specify a value, for example:



Function: Prefix

updated_

5. When you have finished mapping an attribute or table property, click **Save Attribute Mapping** to save the change.

6. Click **Save** to save your mapping changes for the object.

   As you map objects in the source schema to objects in the target schema, the Advanced Multi-Schema Exchanger creates reverse mappings from those target objects to the original source objects. These reverse mappings are used when re-importing transferred data. Click **Reverse** to review the mappings from target objects to source objects.

To remove the mapping for an object,select the item in the summary list and click **Remove Mapping** ⊖.

Dataset mapping

Click **Data Mapping** to list the datasets that will be mapped to stubs when imported because they are not defined in the target schema and do not exist at the target site. You can optionally add SKIP closure rules to the transfer mode to avoid exporting these datasets.

Admin objects and other object mapping

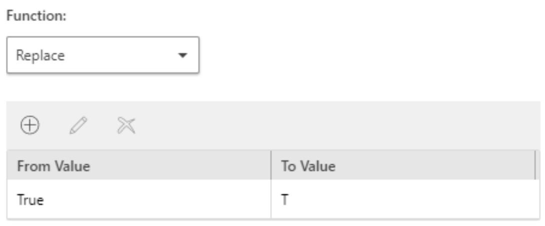
Click **Admin and All Other Object Mapping** to list administrative objects being imported. Review the list and ensure that all errors are resolved before attempting to exchange data.

Map attribute values to new values

Certain attribute values may be represented differently in the source and target schemas. For example, the source schema may require an attribute value of **True**, and the target schema may require a value of **T**. Use the **Replace** function when mapping attributes to change the attribute's value when transferring the data.

1. With the source and target attribute selected in the Advance Multi-Schema Exchanger, set **Function** to "Replace", and click **Add** ⊕ to display the **Add Value Replacement** dialog box.

2. Specify source and target attribute values and click **Add**, for example:


Function: Replace

From Value	To Value
True	T

Mapping attributes with different types and lengths

Siemens Digital Industries Software recommends you always map attributes to attributes having the same types and lengths. By default, Advanced Multi-Schema Exchanger supports only these types of mappings.

For situations where mapping source attributes to target attributes of different types or lengths is required, you can configure Advanced Multi-Schema Exchanger to allow these mappings.

Warning:

Mapping attributes to attributes with different types or lengths may result in truncated or corrupted data. Teamcenter will not import data that does not meet the rules defined in the local schema.

1. Ensure Advanced Multi-Schema Exchanger is closed.

2. Navigate to the following subdirectory of your Teamcenter installation directory (*TC_ROOT*):

   TC_ROOT`\janus\configuration`

3. From that directory, open *userconfiguration.json* in a text editor.

4. To allow attributes to be mapped to attributes of any type, set **allow_map_attributes_with_diff_types** to **true**.

   To allow attributes to be mapped to attributes of any length, set **allow_map_string_attribute_with_diff_length** to **true**.

5. Save *userconfiguration.json*.

Your attribute mapping changes are available the next time you run Advanced Multi-Schema Exchanger.

Validating mapping rules

Validate your project's mapping rules reviews for errors and potential issues before transferring data.

To validate mapping rules, you must be able to log on to Teamcenter as a Teamcenter administrator with DBA privileges. If you cannot log on to Teamcenter this way, you can provide your mapping rules to a Teamcenter administrator for deployment using the steps in Generating mapping rules.

Set up your Teamcenter environment

The Advanced Multi-Schema Exchanger logs on to your Teamcenter environment to validate a project's mapping rules.

1. Ensure the **Forward** tab is selected at the top of the Advanced Multi-Schema Exchanger.

2. In the **Overview** tab, under **Teamcenter Properties**, click **Edit** and supply values for the following fields:

   Teamcenter Root (tcroot)

   The full path to the *TC_ROOT* directory.

   Teamcenter Data (tcdata)

   The full path to the *TC_DATA* directory.

   User Name

   The user id of a Teamcenter administrator who is a member of a group with DBA privileges.

   Password File

   An encrypted password file containing the password of the specified user. Use the **install** command with its **-encryptpwf** option to create an encrypted password file.

   Group

   A user group with DBA privileges.

3. Click **Stop Edits** to save the property settings.

Validate the project's mapping rules

After all objects are mapped and all **Status** levels show 100% completion, you can validate a project's mapping rules.

1. Ensure the **Forward** tab is selected at the top of the Advanced Multi-Schema Exchanger. On the **Validation** tab, select the transfer option set you intend to use to transfer the data.

2. The Advanced Multi-Schema Exchanger validates one *.bcz* file at a time. Ensure **Source File** is set to the *.bcz* file to validate. If your project contains more than one *.bcz* file, validate each file.

3. Click **Validate**. The **Forward** rules in the *.bcz* file are validated and a summary report is displayed.

4. If no issues are found when validating the mapping rules, continue with validating any other *.bcz* files in the project.

   If issues are reported, click on an issue to see additional details and recommendations for addressing the issue.

5. Select the **Reverse** tab at the top of the Advanced Multi-Schema Exchanger and repeat this process to validate the rules for re-importing transferred data. Validating the rules created for re-importing data ensures that mappings such as prefixes and suffixes resolve as expected.

Deploying mapping rules

Deploying mapping rules attaches the rules to the transfer modes used for transferring the data. Use the following steps to deploy your mapping rules. To deploy mapping rules, you must be able to log on to Teamcenter as a Teamcenter administrator with DBA privileges. If you cannot log on to Teamcenter this way, you can provide your mapping rules to a Teamcenter administrator for deployment using the steps in **Generating mapping rules**.

Click **Forward** to use the following steps to attach rules to the transfer mode used to transfer data from the source site to the target site.

If you intend to reimport data transferred to a target site, click **Reverse** and use these same steps to attach the rules to the transfer mode used to transfer data from the target site back to the source site.

1. Ensure that you have **created mapping rules for all objects**, have **validated the mapping rules** in your project, and have addressed any issues. In the **Mapping** tab, if **Completion Status** is 100%, the **Deploy** button is available under **Deployment**.

2. Select the transfer mode set you are using to transfer the data.

3. Click **Deploy**.

   The mapping rules are attached to the transfer mode. In Teamcenter, you can now use the transfer mode to share data.

Detach rules from a transfer mode

Detach rules from a transfer mode as follows:

1. From a Teamcenter command window, enter the following command to determine the name of the rule file attached to the transfer mode:

   ```
   plmxml_tm_edit_xsl -u=Tc-admin-user -p=password
     -g=group -action=list -transfermode=transfer_mode_name
   ```

   Where:

   Tc-admin-user, *password*, *group*

   Login credentials of a Teamcenter administrator with DBA privileges

   transfer_mode_name

   The name of the transfer mode with the attached rules

2. Use the following command to detach the rule file from the transfer mode:

```
plmxml_tm_edit_xsl -u=Tc-admin-user -p=password
  -g=group -action=detach -transfermode=transfer_mode_name
  -xsl_file=rule_file_name
```

Where:

transfer_mode_name

The name of the transfer mode with the attached rules

rule_file_name

The rule name identified in the first step

Generating mapping rules

When creating mapping rules, you must log on to Teamcenter as a Teamcenter administrator with DBA privileges before you can validate and deploy the mapping rules. If you are unable to log on as a Teamcenter administrator with DBA privileges, use the following steps to provide the rules you've created to a Teamcenter administrator for further processing.

1. Ensure you have created mapping rules for all objects as described in **Rule mapping process**. When you have done so, **Completion Status** is 100% for the Briefcase files listed on the **Mapping** tab and the **Generate Mapping Rules** button is available on the **Overview** tab.

2. On the **Overview** tab, ensure **Forward** is selected at the top of the Advanced Multi-Schema Exchanger. Click **Generate Mapping Rules**, select a folder in which to save the mapping rules file, and click **Select Folder**.

   A file named *xrule.csv* is saved in the selected folder. This file contains the forward mapping rules you created.

3. On the **Overview** tab, click **Reverse**. Click **Generate Mapping Rules**, select the folder in which to save the reverse mapping rules file, and click **Select Folder**.

   A file named *rvs_xrule.csv* is saved in the selected folder. This file contains the reverse mapping rules you created.

4. Once you have generated the mapping rules files, provide them to a Teamcenter administrator. The administrator attaches the mapping rules to transfer modes using the steps in **Attaching mapping rules to transfer modes**.

Attaching mapping rules to transfer modes

Typically, you attach mapping rules to transfer modes by **deploying the rules using the Advanced Multi-Schema Exchanger**. However, Teamcenter X users and others without Teamcenter administrator privileges cannot complete the mapping rule validation and deployment steps using the Advanced

Multi-Schema Exchanger. In these cases, Teamcenter administrators with DBA privileges can use the following steps to attach the rules to transfer modes.

Detach any existing mapping rules files

Use the following steps to check if any existing mapping rule files are attached to the transfer mode. If rules are attached, detach the with these steps, also.

1. From a Teamcenter command window, enter the following command to determine the name of any rules file attached to the transfer mode.

   ```
   plmxml_tm_edit_xsl -u=user-id -p=password -g=group
        -action=list -transfermode=transfer_mode_name
   ```

   Where:

   transfer_mode_name

   The name of the transfer mode you are inspecting.

   Any rules files attached to the specified transfer mode are listed.

2. If a rules file with the same name as the file supplied to you by the Teamcenter X user is attached to the transfer mode, enter the following command to detach the rules file.

   ```
   plmxml_tm_edit_xsl -u=user-id -p=password -g=group
        -action=detach -transfermode=transfer_mode_name
        -xsl_file=rule_file_name
   ```

   Where:

   transfer_mode_name

   The name of the transfer mode with the attached rule file.

   rule_file_name

   The name of the rule file identified in the first step of this procedure.

Attach a mapping rules file to a transfer mode

From a Teamcenter command window, attach the rule file to a transfer mode using the **plmxml_tm_edit_xsl** command as follows. (The command must be run for each rules file individually.)

```
plmxml_tm_edit_xsl -u=user-id -p=password -g=group
     -action=attach -transfermode=transfer_mode_name
     -xsl_file=rule_file_name
```

Where:

transfer_mode_name

The name of the transfer mode to which the rule file is to be attached.

rule_file_name

The name of the file containing the mapping rules (*xrule.csv* or *rvs_xrule.csv*).

Once the rules are attached, provide the transfer mode to the user who created the mapping rules for testing and manual validation.

Custom transformation basic syntax

Use the following syntax rules when creating common transformation rules files. See *Custom transformation rule advanced syntax* for syntax supporting complex transformation rules. See *XML transformation tool rule examples* for transformation rule examples.

- Save transformation rules files as comma-delimited (*.csv*) ASCII files in the *<TC_DATA>\xformer* directory.
- Write XML transformation rules in one or multiple *.csv* files. When using multiple files, include the files in a parent file as follows:

```
!include <file_name>.csv
```

- Use relative and absolute paths as desired. The base paths for relative paths are stored using the *PATH* environment variable. In Teamcenter, *<TC_DATA>\xformer* is added to the *PATH* environment variable by default.

Mapping rules

Mapping rules associate source structures to target structures. The pattern for mapping rules is:

```
<primary>|<secondary>|<function>|<condition>
```

Subclause	Usage
`<primary>`	Traverse rule that describes input from the source XML.
`<secondary>`	Traverse rule that describes output to the target XML.
`<function>`	Function rule that describes the transformation from the source value to the target value.
`<command>`	Function rule that filters the input XML available for the mapping rule.

The following symbols are used in mapping rules:

Symbol	Usage
#	Comments
|	Subclause separator

A typical mapping rule follows:

```
# These are comments
Item.object_name|Product.name|concat('tc_',$SOURCE_VALUE)|!
is_empty($SOURCE_VALUE)
```

In this example, the **object_name** attribute of every **Item** is transformed to a **name** attribute on **Product** by prefixing "**tc_**" to the value. The rule is executed if **Item** has that attribute and has a non-empty string value.

Traverse rules

Traverse rules describe a traversal from one node to another node, or from one attribute to another attribute. The rule is constructed using a traverse expression.

The following symbols are used in traverse expressions:

Symbol	Usage
.	Traverse to an attribute from a type.
>	Forward reference to a class.
<	Backward reference to a class.
/	Reference to a subelement
\	Reference to a parent element
*	Wildcard representing "any"
{}	Attribute list on a class.

The following keywords describe elements and attributes:

Keyword	Usage
$MAPPED	All mentioned types or attributes in a rule.
$UNMAPPED	All types or attributes in the input source but not mentioned in a rule.
$REFERENCE	All referenced attributes for the type.

Keyword	Usage
$ELEMENT_NAME	Type name for the type.
$TEXT_CONTENT	Content of the XML element.
$CHILD_ELEMENT	Child element of the XML element.

Traverse rule example:

```
Cpd0DesignSubsetElement<cpd0reuse_element.Cpd0DesignSubsetInstance
    <rlz0realization.Rlz0ModelRealizationMap
```

This traverse rule is constructed using the following traverse expressions:

```
Cpd0DesignSubsetElement<cpd0reuse_element.Cpd0DesignSubsetInstance
Cpd0DesignSubsetInstance<rlz0realization.Rlz0ModelRealizationMap
```

Function rules

Function rules perform the actual transformation to source XML to target values or objects.

See the file *<TC_DATA>\xformer\init.csv* for several sample functions.

The following operators are used in function rules:

Symbol	Usage
+	Addition
-	Subtraction
*	Multiplication
/	Division
=	Equals
!	Logical not
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
!=	Not equal to
==	Logical equal

Symbol	Usage
`and`	Logical and. (Replaces && to avoid symbol conflicts.)
`or`	Logical or
`@`	Returns the first attribute or object based on the current context.
`@@`	Returns all attributes or objects based on the current context.

The following keywords pass values from traverse rules to function rules for transforming:

Keyword	Usage
$SOURCE_VALUE	The first attribute value of the source object.
$SOURCE_VALUES	A list of the source object's attribute values.
$SOURCE_ATTRIBUTE	The source object's attribute name.
$TARGET_END_OBJECT	The last traversed object from the secondary traverse rule.

Function rule example:

```
Product|Item||@Product.type != 'TestPart'
```

In the example, **@** is equivalent to **get_first(@@<traversal>**.

Options

The following default options are available:

Option	Usage
opt_target_root	The root element in the target XML. For TC XML to PLM XML, the default value is PLM XML. For PLM XML to TC XML, the default value is TC XML.
opt_source_separator	Variable-length array (VLA) separator for source input.
opt_target_separator	VLA separator for the target output. For TC XML, the default value is a comma (,). For PLM XML, the default value is a space.

Custom transformation rule advanced syntax

Use the following approaches and syntax to handle less-common transformations.

Function definitions

Define global variables or functions in the rule file with the following form.

```
!define <opt_custom_option> = <value>;
```

The following example defines an accumulator function.

```
!define accumulator = function(){var x = 0; function(){x=x+1}}();
```

Ensure the definition ends with a semi-colon (**;**).

Process ID references

Reading ID references

An element may have attributes referencing other elements. Use the following syntax for reading reference IDs from the input source file.

```
<type>.<referenceAttribute>|<referencedType>.<attribute>|function|
condition
```

The following example reads **IdRef** elements in the source PLM XML. In the example, each **Product.id** value is fetched, concatenated with **#**, and then linked to a corresponding **ProductRevision.masterRef**.

```
!source_id_reference ProductRevision.masterRef|
    Product.id|concat('#',$SOURCE_VALUE)
```

Writing ID references

Use the following syntax for writing reference IDs to the target XML file.

```
<type>.<referenceAttribute>|<referencedType>.<attribute>|function|
condition
```

In the following example, all items referenced with **ItemRevision.items_tag** structures are written to the target XML file. The reference is serialized to a string value with the form **#** + **Item/GsIdentity.elemId**.

```
!target_id_reference ItemRevision.items_tag|
    Item/GsIdentity.elemId|concat( '#', $SOURCE_VALUE)
```

When a reference is addressed by a matching rule, all subsequent rules for the reference are ignored. For example:

```
!target_id_reference A.a|B.id
!target_id_reference A.a|C.puid
```

In the example, the second rule is skipped for **A.a** if the exact object in type **A** can be resolved by the first rule.

Decorating data

To set a value in the target XML that is not dependent on the source XML, write a mapping rule that produces an empty source value in the *.csv* file. For example:

```
|ProductRevision.island_id| $accu()
|ProductRevision.mastrRef>Product.island_id|@ProductRevision.island_id
```

Use a similar approach to produce an empty target value in the *.csv* file:

```
Item.new_name|| concat( 'new_', @Item.name )
```

Process class hierarchies

The class hierarchy rule is used to describe class structures. It specifies the relationships of subtypes to parent types. For example:

```
!source_hierarchy Document\Item
!source_hierarchy DocumentRevision\ItemRevision
!target_hierarchy Product\Structure\StructureBase
```

source_hierarchy specifies the rule for the source XML and **target_hierarchy** specifies the rule for the target XML.

Process class substitutes

In some data models, classes can be substituted for other classes. (For example, **POM_stub** in Teamcenter can substitute for any Teamcenter classes.) Use the following rule to describe this relationship:

```
!source_substitute $MAPPED|POM_stub|POM_stub2|...
!target_substitute $MAPPED|ExternalRef|ExternalRef2|...
```

Object definitions

Some relationships and attributes in the source XML can be used only in function rules. Because they are not defined by a traverse rule, the attributes and relationships are skipped. The following rule defines the attributes and relationships that are used only in function rules.

```
!source_graph_definition ItemRevision.items_tag>Item
```

Data validation

Use transformer rules for data validation. In the following example, the **Cpd0DesignFeature** type is validated and a message is written to the log:

```
!assert_error |
Cpd0DesignFeature.mdl0model_object>Cpd0CollaborativeDesign|
    print_validation_info( $OBJECT , 'CollaborativeDesign is missing')|
    @Cpd0DesignFeature.isExist == 'N'
!assert_error |Cpd0DesignFeature.mdl0revision_id|
    print_validation_info( $OBJECT , 'revision is missing')|
    validate_new_obj_missing_attr($OBJECT, $SOURCE_VALUES)
```

XML output element sequences

XML element sequences can be controlled in the target XML. In the following example, **Header** and **ApplicationRef** are output as the child elements of their parents.

```
!output_element_sequence PLMXML|Header
!output_element_sequence *|ApplicationRef
```

Advance traverse rules

Dynamic type names and attribute names

Type names and attribute names can be dynamically defined based on context with **Function Rule** specified in brackets. For example:

```
# Map Product daynamically decided by Product.type
Product|[ @Product.type ]

# For example map Item.object_name to Product.new_object_name
Item.*|Product.[concat ( 'new_', $SOURCE_ATTRIBUTE)]
```

Filters

Filters can be defined to traverse objects with precise control. Define filters by specifying a **Function Rule** in brackets after any **Traverse Expression**.

Use the following keywords in filters:

Keyword	Usage
$OBJECT	Return every separate object as input from the current traversal.
$OBJECTS	Return all objects as input from the current traversal.

For example:

```
# Map PLMAppUID as ApplicationRef for 1st mapped object at target
$MAPPED<object_uid_str.PLMAppUID|$MAPPED[get_first($OBJECTS)]/
ApplicationRef
```

The resulting behavior is based on returned values as follows.

Value	Behavior
Object or list of objects	The result is taken as the current traversal result. For example, **get_first** in the earlier filter example.
Boolean	If the result is true, the input to the filter is taken as the current traversal result. If the result is false, the input to the filter is not included in the target XML.

String or number	The result is taken as a key to find or create a specific object. For example:
	
	`CD|Product`
	`CD|ProductRevision`
	`CD/GsIdentity|Product/ApplicationRef[0]`
	`CD/GsIdentity|ProductRevision/ApplicationRef[1]`
	`CD/GsIdentity.label|Product/`
	`ApplicationRef[0].label`
	`CD/GsIdentity.label|ProductRevision/`
	`ApplicationRef[1].label|`
	`    concat( $SOURCE_VALUE, '|ProductRevision')`
	
	In this example, **GsIndentity** is mapped to two **ApplicationRef** elements with differing attributes.

Advanced function rules

Using the keyword **function**, you can define a new function based on the transformer function. See the file *<TC_DATA>\xformer\init.csv* for sample functions.

The following keywords are valid when writing custom transformer functions:

Keyword	Usage
true	Boolean true
false	Boolean false
nil	Empty list
...	Variable arguments declaration

Keyword	Usage
$ARGS	Variable arguments used as a list in function definitions
$PATH	Predefined search path when loading files
var	Declares a variable definition
function	Defines a custom function
return	Returns a value from a function
if, **else**, **else if**	Conditional statements
while, **for_each**	Looping statements
break, **continue**	Keyword for flow control

Custom transformation syntax limitations

Text encoding

If types, attributes, or strings contain non-ANSI characters, ensure the files are encoded as UTF-8. If the input XML files are not encoded as UTF-8, multibyte characters may not transform correctly.

Rule execution order

Transformer rules are executed sequentially, except for certain rules that are sorted prior to others. As rule execution results may rely on order, be aware of the following ordering behavior for transformer rules:

Mapping rules

Mapping rules are executed in the following order

1. Rules for object mapping
2. Rules for object relation mapping
3. Rules for attribute mapping

For mapping rules of the same type, rules that contain fewer reference characters (**>** and **<**) are executed before rules containing more reference characters. For rules of the same type and with the same number of reference characters, child types are executed before parent types.

Rules for decorating data

Unlike mapping data, decorating rules directly modify the target XML without a dependency on the source XML.

Rules for reading ID references

Rule evaluation follows the same sequencing behavior as for mapping rules.

Rules for writing ID references

Rule evaluation is based on class hierarchy with the sub type executing first.

Subtraversal scope

Subtraversals support types in source XML. Therefore, subtraversals are supported only on the source side in rule files. In constant and id reference rules, subtraversals are supported on source XML and target XML.

XML transformation tool rule examples

Example: one-to-one mapping

Input

```
<Cpd0Workset item_id="4G_PLMXML_WS01"  object_name="WS01"
        object_type="Cpd0Workset" object_desc="test desc" />
```

Output

```
<Workset catalogueId="4G_PLMXML_WS01" id="id1"
  name="WS01" subType="Cpd0Workset">
    <Description>test desc</Description>
</Workset>
```

Example: reference mapping

Input

```
<Cpd0WorksetRevision elemId="id60" item_revision_id="A"
    object_name="WS01" puid="yUWZc7J3qd$DyB"
    structure_revisions="S6eZc7J3qd$DyB" />
<PSBOMViewRevision bom_view="S2fZc7J3qd$DyB" elemId="id49"
    object_name="4G_PLMXML_WS01/A-View" puid="S6eZc7J3qd$DyB" />
<PSBOMView elemId="id47" object_name="4G_PLMXML_WS01-View"
    puid="S2fZc7J3qd$DyB" view_type="#id24" />
<PSViewType elemId="id24" name="view" puid="zkblwu4OAAgcRA" />
```

Output

```
<View id="id3" name="view">
</View>
<WorksetRevisionView id="id34" viewRef="#id3">
</WorksetRevisionView>
```

Approach

First, map the object.

```
PSViewType|View
Cpd0WorksetRevision|WorksetRevisionView
```

Then, map the source reference and target.

```
Cpd0WorksetRevision.structure_revisions>PSBOMViewRevision.bom_view>
    PSBOMView.view_type>PSViewType|WorksetRevisionView.viewRef>View
```

Example: one-to-N mapping

Input

```
<View id="id3" name="view">
    <ApplicationRef application="Teamcenter" label="zkblwu4OAAgcRA"
    version="zkblwu4OAAgcRA" />
</View>
<WorksetRevisionView id="id34" revisionRef="#id2" viewRef="#id3">
    <ApplicationRef application="Teamcenter" label="yUWZc7J3qd$DyB"
    version="yUWZc7J3qd$DyB" />
</WorksetRevisionView>
```

Output

```
<PSBOMView elemId="id47" object_name="4G_PLMXML_WS01-view"
    puid="S2fZc7J3qd$DyB" />
<PSBOMViewRevision bom_view="S2fZc7J3qd$DyB" elemId="id49"
    object_name="4G_PLMXML_WS01/A-view" puid="S6eZc7J3qd$DyB" />
```

Approach

First, map the object.

```
WorksetRevisionView|PSBOMView
WorksetRevisionView|PSBOMViewRevision
```

Then, create links between the targets.

```
WorksetRevisionView|PSBOMViewRevision.bom_view>PSBOMView
```

Example: N-to-one mapping

Input

```
<View id="id3" name="view">
    <ApplicationRef application="Teamcenter" label="zkblwu4OAAgcRA"
     version="zkblwu4OAAgcRA" />
</View>
<WorksetRevisionView id="id34" revisionRef="#id2" viewRef="#id3">
    <ApplicationRef application="Teamcenter" label="yUWZc7J3qd$DyB"
```

```
        version="yUWZc7J3qd$DyB" />
</WorksetRevisionView>
```

Output

```
<GeneralRelation id="id5" relatedRefs="#id6 #id1"
    subType="Mdl0AttachAttrGroup">
    <ApplicationRef application="Teamcenter" label="SpYZdx3Lqd$DyB"
        version="SpYZdx3Lqd$DyB" />
    <UserData id="id12">
        <UserValue title="mdl0cs_id" value="SpYZdx3Lqd$DyB" />
    </UserData>
</GeneralRelation>
```

Approach

For N-to-one mapping, determine the principle object related to the target object. Any object may be acceptable, but the principle object simplifies the traverse syntax and is clearer to understand when maintaining.

First, map the object.

```
Mdl0AttachAttrGroup|GeneralRelation
```

Then, use a traverse rule to fetch the attribute on the object referenced by the mapped object:

```
Mdl0AttachAttrGroup.relation_type>ImanType.type_name|
GeneralRelation.subType
```

Administrative and organization type mapping example

Some objects are used as administrative and organization types. These elements may be referenced by several other elements. For example, a **user** element can occur several times in the source XML. However, each **user** element has a unique **id** attribute value. Several other elements may reference a particular **user** element by **id** value.

Input

```
<AssociatedForm formRef="#id5" id="id10" role="IMAN_master_form" />
```

Output

```
<ImanType elemId="id18" type_name="IMAN_master_form" />
<ImanRelation elemId="id7" primary_object="QlelxMoGmy_$DB"
    puid="QlflxMoGmy_$DB" relation_type="#id18"
    secondary_object="QpRlxMoGmy_$DB" />
```

Approach

Use a function rule for this mapping.

```
AssociatedForm.role|ImanRelation.relation_type|
    find_or_create_object('ImanType','type_name',$SOURCE_VALUE)
!define find_or_create_object = function ( type, attr, val )
{
    var objs=find_objects(type,val);
    if ( is_empty(objs) )
    {
        var obj = create_object ( type, val );
        add_attribute( obj, attr, val );
        objs = append_left(obj,objs);
    }
    objs;
};
```

Example: lookup table

Use **open_table** to open a table from an existing file. Use **create_table** to create a table when transforming the XML.

Input

```
<Mdl0ApprSrchGeomConstraint elemId="id46" fnd0_tso_flag="N"
    island_id="1" lsd="2017-05-10T20:17:34Z" mdl0ProcessCtrlFlag="14"
    mdl0SourceCriteria="wveF6bA7AABaqA" object_properties="2"
    parent_uid="AGZBAQSgIpvzbA" pid="3124" puid="AeRBAJRsIpvzbA"
    timestamp="gJXlxMdcmy_$DB" type="Mdl0ApprSrchGeomConstraint"/>
```

Output

```
<GeometricConstraint geometricConstraintAction="background" id="id18"
    searchCriteriaAction="foregroundBackground" searchCriteriaRef="#id12"
    subType="Mdl0ApprSrchGeomConstraint">
<ApplicationRef application="Teamcenter" label="AeRBAJRsIpvzbA"
    version="AeRBAJRsIpvzbA" />
</GeometricConstraint>
```

Approach

Prepare the lookup table.

```
   !define GeoConstraintTable = create_table('
   #flag|geometricConstraintAction|searchCriteriaAction
   5|foreground|foreground
   6|background|foreground
   7|foregroundBackground|foreground
   13|foreground|foregroundBackground
   14|background|foregroundBackground
   15|foregroundBackground|foregroundBackground
   ');
```

Then, use a function rule to look up corresponding **ProcessCtrlFlag** values.

```
Mdl0ApprSrchGeomConstraint.mdl0ProcessCtrlFlag|GeometricConstraint.
    {geometricConstraintAction,searchCriteriaAction}|
    lookup_table ( GeoConstraintTable, $SOURCE_VALUE );
```

Tips for customizing PLM XML extensions

Using the PIE_register_user_schema for the export schema mapping

PIE_register_user_schema is designed to alter the export schema mapping of Teamcenter properties to PLM XML element attributes. It is not used in imports. Each PLM XML element has a limited set of attributes available for use. For example, the PLM XML element **Product** has the following defined attributes:

id	**checkoutRefs**	**productId**
name	**subType**	**alternateForRef**
nameRef	**effectivityRefs**	**unitRef**
descriptionTextRef	**releaseStatusRefs**	**designRequired**
attributeRefs	**catalogueId**	**source**
accessRefs	**optionRefs**	**vendorRef**
statusRef	**propertyRefs**	

You cannot map PLM XML attributes not in this list.

For more information about other PLM XML elements and their available attributes, see the PLM XML SDK documentation.

To map Teamcenter custom user properties to the PLM XML file, use a property set. A property set places Teamcenter user properties within a PLM XML **<UserData>...</UserData>** element structure.

Using a TransferMode custom context

When defining a transfer mode, you can leave the **Context** box blank.

Entering the context string used in **PIEUserMethodList_t** or **CUSTOM_PIE_CONTEXT_STRING** uses user-defined method extensions to export or import a specific Teamcenter business object or PLM XML element, respectively.

In the example in this topic, the body of the item revision method extension is empty, so the data is not exported. You export the data using the PLM XML SDK.

Using the XSLT option to translate a PLM XML file

The **XSLT** option provides the option to translate a PLM XML file before import or after export. The following code in the sample code for the **G4PreExportExtension** function sets the XSLT file:

```
PIE_session_set_user_xslt_file(session, "C:\\TCU83\\dev\\xslt\
\prettyprint.xsl");
```

This function sets the XSLT file for import and export. If you want to set the XSLT file for a **TransferMode** object without using customization, use the **plmxml_tm_edit_xsl** command line utility.

Sample XSLT files are available on the internet. Use your preferred internet search engine and search for **XSLT pretty print**.

The XSLT processor is invoked by the PLM SDK.

6. Exporting and importing data as PLM XML

Exporting and importing using PLM XML

PLM XML supports the export and import of Teamcenter objects, such as items, datasets, BOMs, forms, and folders, as well as system data, such as business rules and organization data.

PLM XML is the interoperability mechanism for many products (internal and external). There are a variety of PLM XML representations available to represent the same data. For example, product structure can be represented by normal PLM XML export files from Teamcenter (**ProductView** and **ProductDef**, for example). These files can be used to import the same product structure to another Teamcenter instance. They can also be used by files exported using the **bomwriter** utility (mainly used for Lifecycle Visualization; these files cannot be imported). Before attempting to import a file to Teamcenter, determine the source of the file. If it is exported by another Teamcenter site or an application that matches Teamcenter exported files, it is a good candidate for import to Teamcenter.

PLM XML and the product data management system

A product data management (PDM) system is used to store and retrieve data within a certain context. The context helps define how the data is used, filtered, and transformed. To accomplish this, the PDM system must have the capability to export and import data. In Teamcenter, this is done using the PLM XML import export framework with the various applications written to capitalize on this layer.

One important aspect of data translation is that each translation has a unique purpose and scope. The purpose is associated to a business function and the application-specific interaction with the PDM system. Each instance of a translation is unique and different. A translation must be managed to support its mode of transfer, the traversal of data desired, the filtering of the data to be done and actions to apply to the translation. The PLM XML import export framework supports these unique translations. However, in Teamcenter-to-Teamcenter transfers, do not use PLM XML as a data synchronization mechanism to keep the structure modifications at both sites synchronized. PLM XML export cannot capture all of the edits made on a product structure at the source site. The exported PLM XML file only captures a snapshot of the product structure, so the repeated process of exporting from the source site and importing at the destination site causes unexpected results (the structures at both sites will not be synchronized). In these situations, use Multi-Site Collaboration.

Another important aspect of data translation is managing the mapping between the PDM data and the PLM XML standard entity schema for reading the data in and out. In the PLM XML import export framework, data entities are mapped using a known set of correspondence, where most data is automatically translated by the PLM XML import export module. The PLM XML import export module supports application extensibility for additional data translations. The mapping between PLM XML entities and Teamcenter entities is shown in *PLM XML/Teamcenter object mapping*.

PLM XML and XSLT

PLM XML/TC XML Export Import Administration references XSLT (eXtensible Stylesheet Language Transformation) files that describe how to transfer the source tree or data structure of an XML document into the result tree for a new XML document, which may be completely different in structure, and can be stored with the transfer mode objects and applied as a post-export or pre-import operation.

Note:

The PLM XML framework does not support XSLT file validation; therefore, it is your responsibility to ensure valid XSLT format.

Transfer mode objects

In PLM XML/TC XML Export Import Administration, you create transfer mode objects that define the import or export context by applying closure (traversal) rules, filter rules, action rules, and property set rules to the input or output of objects and system data. These rules can be a static set stored in the database or they can be applied only to a specific session. Rules stored in the database can be imported and exported.

Exporting BOMs using PLM XML

When you export Teamcenter data as PLM XML, you can export only a single BOM. Do not configure multiple BOM structures for export as PLM XML.

Multiuser environments

In a multiuser environment, when multiple users initiate import on the same data simultaneously (for example, multiple users updating the same assembly), only one is successful. The other users get a locking message (objects are locked by different user in different session).

PLM XML and Multi-Site Collaboration

PLM XML cannot be used to share organization objects that are shared through global organization (Multi-Site).

Asynchronous PLM XML export

You can export objects into Teamcenter asynchronously through the rich client. An asynchronous export uses the **plmxml_import_export** translator to run the **plmxml_export** utility in the background.

To enable the **plmxml_import_export** translator, see the *Dispatcher —Deployment and Administration*.

Using Teamcenter to export and import data with PLM XML

Once you create the transfer mode object in PLM XML/TC XML Export Import Administration, you can use the rich client, Active Workspace, or a command line utility to import or export data. Be aware of the following items.

- If you are importing a named reference and the named reference XML file name and subfolder name are different, import it with the **plmxml_import** utility instead of the rich client. If the file name and the subfolder name are the same, you can use either the utility or the rich client.
- When exporting PLM XML, choose to use a transfer mode of interest.
- When exporting 4G PLM XML, first export the data to TC XML, transform it into PLM XML, and then export it using the PLM XML transfer mode. Because 4G PLM XML is based on the Teamcenter Import Export (TIE) framework, using this process internally leverages transfer option set capabilities
- Consider using the TIEPLMXMLExportInternal_TM and TIEPLMXMLImportInternal_TM TCXML transfer modes. For example, if you have PLMXML export transfer mode XYZ. Whenever you use that TM for export, the export internal TM will always be included. If there is a Transfer Option Set attached to TIEPLMXMLExportInternal_TM whose name is prefixed with XYZ, that Transfer option set is used.
- If you need to create a new PLMXML TM, you may need to create a transfer option set and attach the TOS to TIEPLMXMLExportInternal_TM or TIEPLMXMLImportInternal_TM. This is often the case when creating the TM by copying another one. When doing so, you need to verify that the copied TM has a corresponding transfer option set. That relationship is by name. For example, TM 4GDPIEDataExportTcVis has transfer option set 4GDPIEDataExportTcVis_TOS.

If the transfer mode object does not exist, you can create it with the PLM XML/TC XML Export Import Administration application, as described in *Create transfer mode objects*. You can also use programming (*Teamcenter Server Customization* and the *Integration Toolkit Function Reference*) to create transfer mode objects.

Export to a PLM XML file

1. Select the objects to be exported.

   You can export only a single BOM. Do not configure multiple BOM structures for export as PLM XML.

2. Choose **Tools→Export→To PLMXML**.

   The system displays the **PLM XML Export** dialog box.

3. In the **Export Directory** box, select the directory in which to place the PLM XML file. Click **Browse** to locate the directory.

4. Specify the name of the export file in the **Export Filename** box.

5. Choose a transfer mode from the **Transfer Mode Name** list.

6. (Optional) Click **Select Languages** to display the **Language Selection** dialog box for the languages configured at your site.

    - The **Available Languages** box lists the languages you can select as additional languages for export.

    - The **Select Languages** box lists the preconfigured languages for the selected transfer mode.

    - To select additional languages for export, follow these steps:

        a. Select a language in the **Available Languages** box.

        b. Click the **Add** button ➕.

            - The selected language is added to the **Selected Languages** box.

            - To reorder the languages in the **Selected Languages** box, select a language and use the move ▲ ▼ buttons as required.

        c. Click **OK** or **Apply**.

            The languages selected for export are displayed in the **Languages** box.

            Note:

            If you change the transfer mode, the **Languages** entries remains unchanged. You can use the **Language Selection** dialog box again to add or remove more languages.

7. (Optional) Select a revision rule to configure the assembly to be exported for the selected root object.

    Note:

    If you select multiple root objects, no revision rule is applied, so only the selected root objects, not the structure, are exported.

8. (Optional) Select **Open PLM XML File** to view the file when the export operation is complete.

9. (Optional) Select the **Perform Export In Background** check box to perform an asynchronous export.

10. Click **OK** to export the objects in the **Object List** list and close the dialog box.

When the export completes, a dialog box displays.

If the export was successful, you can click **Yes** in the dialog box to view the log file for the export.

If there were errors during the export, the dialog box lists the errors. Click **Yes** to view information about the errors that occurred.

Note:

Exports that completed without errors in past versions may now show errors. Previously, unless the **PLMXML_log_file_content** preference was set to **detailed**, errors did not display. The detailed error information now displays in the log file by default. Setting the **PLMXML_log_file_content** preference to **detailed** produces detailed information of all objects (those successfully exported and those exported with errors).

Import data from a PLM XML file

1. Choose **Tools→Import→From PLMXML**.

2. In the **PLM XML** dialog box, click the **Browse** button ••• to the right of the **Importing Object** box.

   The system displays the **Select Object** dialog box.

3. Select **PLM XML File (.xml)** from the **Files of type** list.

4. Navigate to the directory containing the file, and select the file.

5. Click the **Select** button.

6. Select the transfer mode to be used to configure the import operation.

7. (Optional) Create or select an incremental change object into which BOM components contained in the XML file will be imported.

   Importing BOM components into an incremental change object allows you to view changes in Structure Manager prior to making them effective.

   Note:

   BOM components imported into an existing incremental change object are appended to the list of incremental change components.

8. Click **Apply** or **OK**.

   The system imports the data in to your **Newstuff** folder.

Note:

If object names or IDs are encountered that exceed the character limit for those attributes in Teamcenter, the name and/or ID is truncated when imported in to Teamcenter.

A. Troubleshooting issues when importing and exporting PLM XML

Enable environment variables for debugging

Enable the following environment variables to provide additional debug information. The setting of **PIEDBG_TOCONSOL** or **PIEDBG_TOLOG** control the location of where the information is written.

Variable	Value	Purpose
PIEDBG_SESSIONTRACE	**1**	Writes a trace of session activities.
PIEDBG_BOMEXPORT	**1**, **2**, or **3**	Writes information about the assembly export. The larger the number, the more information is written.
PIEDBG_ADDTOPARENT	**1**	Writes information about the add-to-parent call sequence (parent-child relationship).
PIEDBG_TRAVERS	**1**, **2**, or **3**	**1** – Writes information about each top-level entity being translated. **2** – Writes information about every major step in the process and traversal loop. **3** – Writes information plus dumps the message structure in the process.

Review the PLM XML log file

Review the PLM XML log file. Note that this is different from the **syslog** file.

Following is a sample PLM XML log file. An actual log file also contains information about any errors that occur during the import or export.

```
****************************
   PLMXML Log File Header
   ----------------------
            IMPORT
****************************
```

```
LOG-FileName is: path-of-generated-log-file
XML-FileName is:  path-of-xml-file
XSLT-FileName is:
OverWrite-ExistingData is:  0
CheckOutObjects-UponExport is:  0
CheckFor-ExportPrivilege is:  1
****************************
///// Teamcenter Import Summary BEGIN: Processed the following PLMXML
Elements
  Element:: DataSet                                             Count:: 1
  Element:: ExternalFile                                        Count:: 1
///// Teamcenter Import Summary: Processed the following Teamcenter
Objects
 TCObject:: DXF                                                 Count:: 1
 TCObject:: ImanFile                                            Count:: 1
///// Teamcenter Import Summary END: /////
```

Enable detailed logging

Enable detailed logging (set the **PLMXML_log_file_content** preference to **detailed**). The PLM XML log file contains additional information, such as details on each object and a Teamcenter-to-PLM XML object comparison. For example:

```
****************************
   PLMXML Log File Header
   ----------------------
            EXPORT
****************************
LOG-FileName is: path-of-generated-log-file
XML-FileName is:  path-of-xml-file
XSLT-FileName is:
OverWrite-ExistingData is:  0
CheckOutObjects-UponExport is:  0
CheckFor-ExportPrivilege is:  1
****************************
PLMXML_sdk_threshold is:  100
******************************************************
****************************
Teamcenter Object ====>
Processed Object [wVChlmCCzmZcYC] of type [POM_imc]
IMC-999888777
Exported As ====>
Processed Element [Site] Id [#id5]
Error: None
****************************
****************************
Teamcenter Object ====>
Processed Object [SkBlTDvQzmZcYC] of type [ItemRevision]
```

```
Exported As ====>
Processed Element [ProductRevision] Id [#id2]
Error: None
****************************
****************************
Teamcenter Object ====>
Processed Object [iIDlTDvQzmZcYC] of type [Text]
Exported As ====>
Processed Element [DataSet] Id [#id7]
Error: None
****************************
****************************
Teamcenter Object ====>
Processed Object [iMPlTDvQzmZcYC] of type [ImanFile]
Exported As ====>
Processed Element [ExternalFile] Id [#id9]
Error: None
****************************
****************************
Teamcenter Object ====>
Processed Object [SgIlTDvQzmZcYC] of type [Item]
Exported As ====>
Processed Element [Product] Id [#id10]
Error: None
****************************
=====  END OF (DETAILED OBJECT TRANSLATION) =====
///// Teamcenter Export Summary BEGIN: Processed the following
Teamcenter
      Objects
 TCObject:: ImanFile                                       Count:: 1
 TCObject:: Item                                           Count:: 1
 TCObject:: ItemRevision                                   Count:: 1
 TCObject:: POM_imc                                        Count:: 1
 TCObject:: Text                                           Count:: 1
///// Teamcenter Export Summary: Generated the following PLMXML Elements
 Element:: DataSet                                        Count:: 1
 Element:: ExternalFile                                   Count:: 1
 Element:: Product                                        Count:: 1
 Element:: ProductRevision                                Count:: 1
 Element:: Site                                           Count:: 1
///// Teamcenter Export Summary END: /////
```

Set PLM XML logging to different levels

Set PLM XML logging to different levels, depending on the information you want to capture. To minimize logging, which improves speed on large translations, set the **PLMXML_log_file_content** preference to **basic**. To get closure rule analysis, which can help you tune your closure rules for better performance, set the **PLMXML_log_file_content** preference to **detailed**. The following figure shows an example closure rule analysis.

```
 /////   BEGIN – CLOSURE RULE CLAUSE ANALYSIS REPORT
Transfermode name:     incremental_import
Closure rule name is:     readOccurrenceIncremental_Import
Clause-1: CLASS: Document: CLASS: Form: ?: *: SKIP:
Clause-2: CLASS: *: CLASS: ProductView: ?: *: PROCESS:
Clause-3: CLASS: ProductDef: CLASS: InstanceGraph: ?: *: SKIP:
Clause-4: CLASS: *: CLASS: *: ?: *: DO:
Clause-5: CLASS: *: CLASS: *: PROPERTY: *: DO:
//////  BEGIN – Specific Closure Rule Clause Evaluation
Possible Culprit:  PROPERTY * Found in the following clauses of the
closure
  rule
    Found in 5 clause
Possible Culprit:  Primary CLASS * Found in the following clauses of the
  closure rule
    Found in 2 clause
    Found in 4 clause
    Found in 5 clause
Possible Culprit:  Primary TYPE * Found in the following clauses of the
  closure rule
       (none)
Possible Culprit:  Secondary CLASS * Found in the following clauses of
the
  closure rule
    Found in 4 clause
    Found in 5 clause
/////   END - Specific Closure Rule Clause Evaluation
/////   END – CLOSURE RULE CLAUSE ANALYSIS REPORT
```

The default setting for the **PLMXML_log_file_content** preference is **summary**, which provides more information than **basic**, but less than **detailed**.

Review the syslog file for EntireExport or EntireImport

Review the **syslog** file for statements with **EntireExport** or **EntireImport**. For example:

```
PIESession:ProcessExport/EntireExport Metrics - the PLMXML export
operation
has been completed, here are the performance metrics:
 SM Memory Usage(bytes):          xxx
 Total Memory Usage(bytes):       xxx
 CPU time(seconds):               xxx
 Real time(seconds):              xxx
 DB Trips:                        xxx
```

Enable additional logging

Enable additional logging from Teamcenter by setting the Teamcenter logging environment variables.

Enable additional logging from the PLM XML Import Export module by setting the following environment variables.

Variable	Value	Purpose
PIEDBG_TOCONSOL	**1**	Writes debug information to the console.
PIEDBG_TOLOG	**1**	Writes debug information to the PIE log file.
PIEDBG_TOSYSLOG	**1**	Writes the call sequence to the **syslog** file (helpful in crash-like situations).

Troubleshooting PLM XML exports and imports

Following are specific scenarios that you may encounter while using PLM XML/TC XML Export Import Administration.

Issue	Possible cause	Possible solution
PLM XML file not generated/PLM XML file generated but physical files not generated	• FMS not set up properly. • User does not have write permissions to the destination directory.	• Verify by uploading a physical file to a standalone dataset from the rich client. This ensures that the basic FMS/volume settings are correct. • Correct the FMS settings. • Perform the export/import from command line using **plmxml_export** or **plmxml_import**. This eliminates issues with the transient volume/FMS settings. If this works, resolve FMS setting issues. • Review the **syslog** file to see if the PIE export completed on the server side. • Check the transient volume to see if the server completed the generation.
Intended object not exported	• Object not supported in PLM XML schema.	• Verify the PLM XML **entity mapping** (schema) and Teamcenter PIE implementation for supported objects.

Issue	Possible cause	Possible solution
	• Object supported in PLM XML schema but not supported in Teamcenter PLM XML/TC XML Export Import Administration. • Object supported PLM XML schema but closure rules are not traversing to the intended object. • Closure rules just traverse instead of process on the intended object. • Closure rules in place but **FilterRule** is filtering out the object. • Everything is in place but there is an error during the export of the intended object.	• Verify that the closure rules are in place and their directives are **Process** or **Traverse+Process**. • Review the PLM XML export log file in detailed mode to see if there are errors during the export of the intended object (search for **ERROR** in the log file).
Intended object exported but intended property not exported	• The intended property is not exported by default in the PLM XML schema (for example, **object_string** on **ItemRevision**). • The intended property is not part of the property set used for export. • There is a functional error in writing the property value to the PLM XML file.	• If the property is part of the PLM XML schema, no property set is required. Teamcenter PLM XML/TC XML Export Import Administration supports it. • If the property is not part of the PLM XML schema, you must specify a property set clause to export the intended property. • Review the PLM XML export log file in detailed mode to see if there are errors during the export of the intended object (search for **ERROR** in the log file).

Issue	Possible cause	Possible solution
Import failure	• When importing from the rich client, the PLM XML file and named references are transferred to the transient volume on the server before initiating the import. There may be issues with FMS. • The PLM XML file is not well formed and cannot be loaded. • There is a functional error with XML element processing. • The object types/ data model items are not present at the importing site. • The item ID of items being imported already exist or are owned by a different user.	• Make sure the FMS settings are correct and the PLM XML file is successfully transferred to the transient volume. • Perform the export/import from command line using **plmxml_export** or **plmxml_import**. This eliminates issues with the transient volume/FMS settings. If this works, resolve FMS setting issues. • Review the PLM XML import log file and ensure there are no errors (search for **ERROR** in the log file).
No objects found in **Home** folder/selected folder after import	• Import may have failed. • PLM XML import does not copy/import the objects to **Home** or selected folder. This is expected behavior.	• Review the PLM XML import log file and ensure there are no errors (search for **ERROR** in the log file). • Perform a general search in Teamcenter to ensure that the objects are imported but hanging in database. • Set the **PLMXML_put_objects_in_newstuff_on_import** user preference to **true** to copy the root objects of the importing file (pointed to by **traverseRootRefs**) to a new folder under the user's **Newstuff** folder. This new folder's name is the same as the XML file name that is imported.

Issue	Possible cause	Possible solution
Physical files not imported	• FMS is not configured properly. • Functional errors in uploading the physical file (for example, a binary file uploaded to a text dataset).	• Try uploading a physical file to ensure that the basic FMS settings are correct. • Perform the export/import from command line using **plmxml_export** or **plmxml_import**. This eliminates issues with the transient volume/FMS settings. If this works, resolve FMS setting issues. • Review the PLM XML import log file and ensure there are no errors (search for “ERROR” in the log file). Understand the summary.
Specific object in file not imported	• The intended object may not exist in the exported **.xml** file (which makes it an export issue).	• Use the **incremental_import** transfer mode to import everything from the **.xml** file. **ConfiguredDataImportDefault** imports only those objects that are related to other objects and based on the closure rules. • Review the PLM XML import log file and ensure there are no errors (search for **ERROR** in the log file).
Specific property on object not set (exists in **.xml** file)	• The intended property of the object may not exist in the exported **.xml** file (which makes it an export issue).	• Use **incremental_import** transfer mode to import everything from the **.xml** file. **ConfiguredDataImportDefault** imports only those objects that are related to other objects and based on the closure rules. • Review the PLM XML import log file and ensure there are no errors (search for **ERROR** in the log file).
Improper importing of **date** type property	Importing a **date** type property from a PLM XML file honors the date format defined in the **timelocal_locale.xml** file. The date format in the PLM XML file does not match the date format defined in the **timelocal_locale.xml** file.	• Edit the **DefaultDateFormat** key in the text server **.xml** file (typically pointed to by the **TC_MSG_ROOT** environment variable). Note that changing the **DefaultDateFormat** key impacts the date format for all of Teamcenter.

Issue	Possible cause	Possible solution
Objects fail to import and one of the following errors is reported: `Unable to find a property with name` *object/attr* or `Xml attribute [name =` *attr*`] not found in database for [`*object*`] element`	When importing XML containing attributes not defined at the target site, the item with the undefined attribute (and possibly other items) will not be imported. • When importing high-level TC XML, the item with the undefined attribute will not be imported. • When importing low-level TC XML, the item with the undefined attribute will not be imported, and any other items on the island will not be imported.	Define the attribute at the target site.
Exporting wrong version of dataset	• Exporting a dataset while it is being simultaneously modified by a different user may result in the wrong data being exported.	• Create a custom filter rule for dataset class objects, where it refreshes the object before it is exported. For example: `CLASS Dataset userRefreshObjectFilter` Then, add the filter rule to the transfer mode used for exporting.
Exporting wrong version of object	• Exporting an object while it is being simultaneously modified in a different session may result in the wrong data being exported.	• Use the ExportLatestObjectFilter filter rule to refresh objects before exporting. For example: `CLASS.Dataset:ExportLatestObjectFilter` Then, add the filter rule to the transfer mode used for exporting. This can be used to refresh any POM object before the export.

Issue	Possible cause	Possible solution
While processing a **CfgAttachmentLine** object, the export completed with the following error: `ERROR 203477 The source object of the attachment line is "PseudoFolder", which is not a persistent object.`	A closure rule clause is instructing the system to process all attachment lines, but one of the attachment lines corresponds to **PseudoFolder**, which is not a persistent object. So the underlying attachments cannot be processed.	Since this is a runtime object, the actual attachment object may have already been processed by a different clause that has persistent data model traversal. To avoid this error, add the following clause just above the existing **bl_attachments** clause in the closure rule. `CLASS.ImanItemLine : CLASS.* : PROPERTY.bl_attachments : PROCESS+TRAVERSE : SECONDARY.al_source_class != “PseudoFolder”`
While the object in the XML file imports correctly, the import completed with the following error: `ERROR 38015 Unable to find a property. An error has occurred during the processing of some objects.`	The import XML file with a **UserData** element that contains business object attributes that are not COTS attributes (for example, **UnitOfMeasure**).	User can either import the data using designed transfer mode (for example, for **UnitOfMeasure** use **ConfiguredDataImportDefault**), or ignore the error.
Intended Microsoft Office template imported but intended styles, macro and header/footer not imported.	• PLM XML import does not support import of Microsoft office styles, macro or header/footer.	Use the Teamcenter rich client command **Tools→Import→Templates→Specification Template**