TEAMCENTER

Teamcenter Data Exchange

Teamcenter 2412

SIEMENS

About Siemens Digital Industries Software

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

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

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

Contents

Loading bulk data into Teamcenter from other sources

Copying product data to test environments

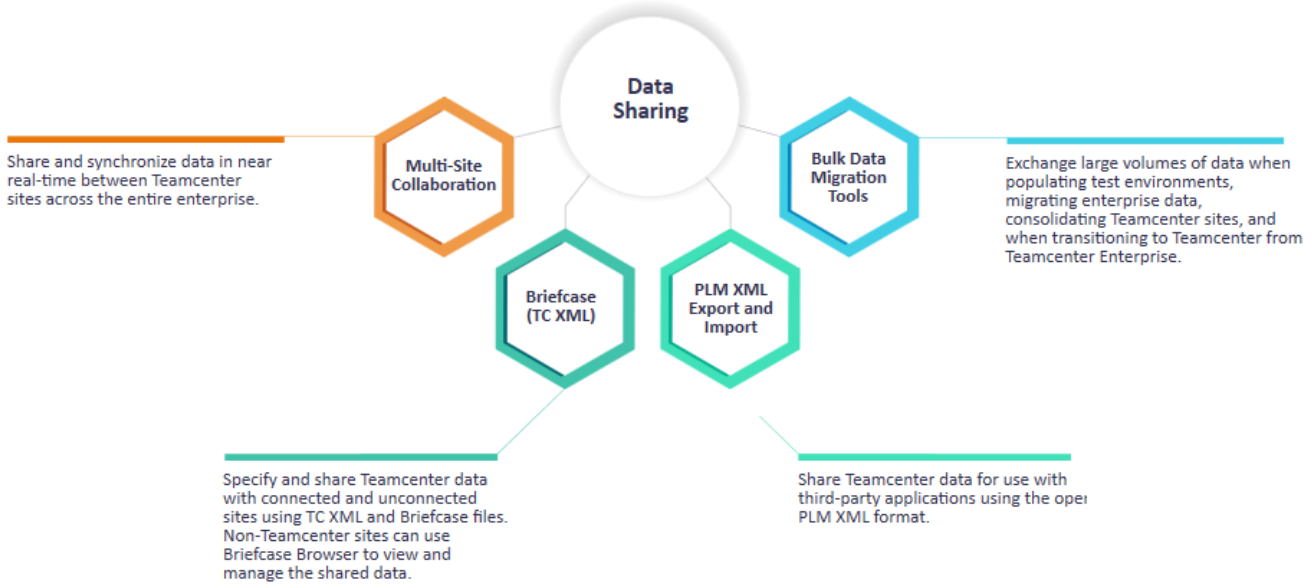
1. Exchanging Teamcenter data with other sites

Sharing Teamcenter data

Collaborating on product design often involves sharing Teamcenter data with other organizations and sites. You may have one or more of the following needs:

- Several groups within your company rely on having access to a project's latest updates as they work on different aspects of the project. These groups may be on site or in other parts of the world.
- Outside organizations, such as suppliers supporting OEMs, have similar data requirements. The suppliers may need a mix of read and write access to the data as they may be responsible for the development of certain parts of the project.
- Some groups may be downstream customers with applications that leverage your data as part of their business processes.
- You may be combining or archiving existing sites as the company structure changes or projects are retired.

Teamcenter offers several solutions crafted to meet these different data exchange needs. *Teamcenter Data Exchange* **introduces each of these solutions**, provides **common configuration information**, and directs you to further information on each data sharing solution.



Where do I go from here?

Administrator	
Configuring and exchanging data using TC XML.	See **Data exchange concepts**.
Migrating large volumes of data into Teamcenter using TC XML.	See **Loading bulk data**.
Populating test environments.	See **Copying product data**.
Creating and managing closure rules, filter rules, and transfer mode objects for exchanging TC XML and PLM XML.	See Working with transfer mode objects.
Exchanging data as PLM XML.	See Exporting and importing using PLM XML.
Administering and using Multi-Site Collaboration	See Configuring and administering Multi-Site Collaboration.
Consolidating and retiring Teamcenter sites.	See Site consolidation.
Business User	
Learn more about each data sharing solution	See **Options for sharing data**.
Package and share data using Briefcase	See **Exchanging Briefcase files**.
Share data in your Multi-Site federation	See Using Multi-Site Collaboration.
View Briefcase files using Briefcase Browser	See Using Briefcase Browser.

Options for sharing data

Teamcenter data sharing solutions address many needs. For example:

Data sharing need	Typical source system	Typical target system	Data sharing solution
Collaborating with others sites using the same or different schemas. Sites may or may not be connected by a network.	Teamcenter	Teamcenter	Briefcase
Exchanging data with suppliers who may need to update the shared data.	Teamcenter	Briefcase Browser	Briefcase and Briefcase Browser
Accessing and regularly updating data across multiple sites within the same or different companies. The sites are all connected by a network.	Teamcenter	Teamcenter	Multi-Site Collaboration
Populating a test environment with production data.	Teamcenter	Teamcenter	Bulk extract and bulk load utilities
Sharing data with third-party applications	Teamcenter	Third-party application	PLM XML export and import

Data sharing need	Typical source system	Typical target system	Data sharing solution
Migrating large amounts of legacy or Teamcenter Enterprise data.	Legacy or Teamcenter Enterprise	Teamcenter	Bulk load and csv2tcxml utilities
Retiring and consolidating multiple Teamcenter sites.	Teamcenter	Teamcenter	Site consolidation

Briefcase

With Briefcase files, you can share Teamcenter data with sites that are not using Multi-Site Collaboration or Teamcenter Integration Framework to share data. You can exchange Briefcase files with other Teamcenter sites or with non-Teamcenter (unmanaged) sites. Unmanaged sites can use Briefcase Browser to work with Teamcenter Briefcase files.

See **Exchanging Briefcase files** for details on configuring, packaging, and exchanging Briefcase files.

Briefcase Browser

Sites without Teamcenter access (for example, suppliers not running Teamcenter) can use Briefcase Browser to work with Briefcase files. Aside from viewing the structure and objects in Briefcase files, Briefcase Browser lets unmanaged sites perform tasks such as:

- Adding new local supplier parts
- Changing existing supplier parts in structures
- Adding a local supplier top assembly to an OEM supplied structure
- Completing CAD assemblies

See *Briefcase Browser* for details on installing configuring, and using Briefcase Browser.

Multi-Site Collaboration

Using Multi-Site Collaboration, you can share Teamcenter objects between multiple connected sites, providing semi automated real time data synchronization across the entire enterprise. Objects are owned by a single site, with copies (replicas) available at other (remote) sites. Remote sites can be granted the ability to update objects they do not own, with Multi-Site maintaining data accessibility integrity.

See *Multi-Site Collaboration* for more information.

Bulk extract and load utilities for populating test environments with data

Teamcenter provides tools to extract large amounts of data from one Teamcenter environment to copy to another. With these tools, you can populate testing environments, test Teamcenter upgrades with real-world data, troubleshoot issues, and more.

See **Extracting and loading bulk production data** for more information on creating test environments and populating them with production data.

PLM XML export and import

Use PLM XML export and import to share data in a standard format with other organizations or with third-party applications. With PLM XML, you can tightly control the scope, versions, and formats of the data you share.

See *PLM XML/TC XML Export Import Administration* for information on sharing Teamcenter data using PLM XML.

Bulk load utilities for migrating bulk amounts of legacy or Teamcenter Enterprise data

When migrating to Teamcenter from Teamcenter Enterprise or other applications, you made need to import large amounts of data into your new environment. Use Teamcenter's TcXML import utility and CSV to TC XML converter to quickly import data by bypassing business rules.

See **Understanding the process of loading bulk data** for more information on bulk loading data.

Site consolidation

As your business needs change, you may need to retire Teamcenter sites or consolidate multiple sites. This site consolidation process requires substantial planning and uses several Siemens Digital Industries Software tools.

See *Site Consolidation* for details for preparing for and performing site consolidations.

Data sharing concepts

Common terms and components

Following are common terms and concepts used when sharing Teamcenter data.

- **Site**

  A site () is a Teamcenter installation or manufacturer or supplier that uses some other method to manage their product data.

- **Export**

  The data sharing function used to send data to a remote site. While exporting the data, you can either transfer the data with ownership or transfer the data for reference. If you transfer the data for reference, the ownership is retained by the exporting site and a replica is exported to the remote site.

- **Import**

  When you export data from a remote site to a Teamcenter site, the site imports the data automatically.

- **Replica**

  Replication is the act of creating an exact copy of an object, known as a replica, at a specific site. Replicas are objects that are owned by a remote site. Whenever a primary object is modified, you must update the replicas by synchronizing them to the primary object. A replica is a nonwriteable object and cannot be updated except by the owning site. When an object is replicated, you cannot delete the primary object unless all the replicas are deleted.

  A replica is represented by a symbol with two green dots.

- **Synchronization**

  When a primary object is replicated at other sites, you must update the replicas whenever the primary object is modified. The process of updating replicas is referred to as *synchronization*.

  Users can manual synchronize data by re-exporting or re-importing an object using the same transfer formula. Multi-Site Collaboration provides automatic synchronization. Teamcenter Integration Framework supports automatic synchronization for push cases and is initiated by making a call to transfer updated objects.

  > Caution:
  >
  > When manually synchronizing a replica, both the owning site and replica site must be online to receive replica deletion notification.

- **Closure rule clause**

  A closure rule clause () controls the scope of the data transfer. It defines the rules for gathering objects during the export action. Closure rules specify how the data structure is traversed by specifying the relationships of interest and the actions to occur when these relationships are encountered.

  Teamcenter also has many internal closure rules that handle out-of-the-box data model traversal. These are normally imported during installation or upgrade. They must not be changed. For instance, the inclusion of the BOM and Absolute Occurrence network is handled by internal closure rules.

- **Transfer mode**

  A transfer mode ( ) is a logical grouping of closure rule clauses. You select a transfer mode when creating closure rule clauses. Transfer modes allow users to export and import data by knowing only the transfer mode name that they must use, for example, **ToSiteA** or **FromSiteB**.

- **Transfer option**

  Specifies the different options you can choose to use when Teamcenter transfers an object. The different transfer options are:

  - **Include All Versions**
  - **Latest Versions**
  - **Selected Version**
  - **Include Entire BOM**
  - **Transfer Top Assembly**

- **Transfer option set**

  A transfer option set (TOS) ( ) is a stored set of transfer options used for remote data export. A transfer option set displays all of the unique options in the closure rule conditional clauses for the selected transfer mode. The **Description** column of the **TransferOptionSet** pane in the PLM XML/TC XML Export Import Administration application describes the purpose of the options included in the standard Teamcenter transfer option sets.

- **Factor**

  A factor is a logical concept defined by a set of objects at the exporting site that map to a similar logical concept defined by a different set of objects at the importing site. You define a factor using a closure rule. Closure rules help optimize data sharing performance.

- **File Management System (FMS) file transfer**

  File Management System (FMS) is used for transferring files by geographically separated work groups whose sites are connected by wide area networks (WANs). It also allows access to shared documents in local area network caches, helps avoid round-trips across high-latency WAN networks to reach sites.

  FMS uses a unique file identifier to determine when to retrieve a file from its local cache rather than from across a network from a remote site. Binary and text files have a different GUIDs if they are replicated. However, if you change the file content by one bit or change its language encoding, the system creates a new file GUID to describe the new contents of the file.

- **Single sign-on (SSO)**

  Security Services allows a user to sign on one time for access to multiple Teamcenter products.

- **Export protection**

  Export protection specifies export-protected objects are shared as stubs.

- **Replica deletion**

  The replica deletion capability deletes the export record of an object on the owning site when a replica is deleted on the remote site.

- **Stub replication**

  The stub replication capability ensures that when a primary object replica is sent to another site, the exporter creates an export record for this new site and tags it as **stubbed**.

- **International Traffic in Arms Regulations (ITAR) license**

  Teamcenter provides support for enforcing policies of ITAR (International Traffic in Arms Regulations) to control dissemination of certain types of information through ITAR licenses that you can attach to workspace objects. If an ITAR license exists at the destination site, the **license_id** attribute associated with the workspace object can be imported.

- **Audit monitoring**

  The integration framework provides an audit log of transactions during transfers between Teamcenter Enterprise and Teamcenter that is maintained in the integration framework database. The following transactions are logged:

  - Get objects for ownership transfer.
  - Transfer ownership to site.
  - Update ownership transfer to source site.

- **Organizational/license object audit logs**

  Data sharing uses the TC XML import and export (TIE) functionality to transfer objects between sites. TIE does not support replicating audit record business objects. Do not attempt to replicate the following objects between sites:

  - **Fnd0LicenseChangeAudit**
  - **Fnd0LicenseExportAudit**
  - **Fnd0OrganizationAudit**

- **Administrative objects**

The TC XML framework processes Teamcenter administrative objects differently than other objects. For example, if an administrative object identified in the TC XML file is not found at the importing site, the import fails whether or not the exporting user selected the **Continue On Error** option. Objects of the following classes are considered administrative objects by the framework:

- **ADA_License**
- **ClosureRule**
- **Condition**
- **CondValAgent**
- **CondValAgentRevision**
- **CondValData**
- **CondValDataRevision**
- **Filter**
- **Fnd0AlgebraicFormula**
- **FunctionalityRule**
- **Group**
- **GroupMember**
- **IdContext**
- **ImanType**
- **NoteType**
- **PIEActionRule**
- **POM_imc**
- **PropertySet**
- **PSOccurrenceType**
- **PSViewType**
- **ResourcePool**
- **RevisionRule**
- **Role**
- **TransferMode**
- **TransferOptionSet**
- **TC_Project**
- **UnitOfMeasure**
- **User**
- **VerificationRule**

Exchanging data when upgrading Teamcenter and performing other transitions

Exchanging data is necessary when you are upgrading multiple Teamcenter sites that have disparate data and operational models. High-level considerations for the upgrade process are:

- Isolating customizations at each site.
- Standardizing (harmonizing) shared customizations.
- Packaging and distributing customizations to all sites.

- Standardizing organizations, access rules, naming and deep copy rules, work processes, and so forth.

  Site standardization can be joined with site consolidation.

- Upgrading sites to an upgrade supported Teamcenter version as interim step.
- Enabling target sites with the harmonized data model.
- Populating new sites using bulk loading tools, transforming or mapping data as needed.

You can also use the integration framework components to transition third-party or custom systems, such as non-Siemens Digital Industries Software PLM or PDM systems, to Teamcenter.

> Note:
>
> Carefully consider your specific requirements during the planning phase of a transition. The following descriptions provide very high-level considerations. Contact Siemens Digital Industries Software services for assistance in determining the best approach for your companies requirements.

High-level considerations for the transition process include:

- Adding a custom integration framework connector.
- Providing methods for export, import, and confirm.
- Leveraging connectors developed by Siemens Digital Industries Software partners for third-party competitive systems, such as, ENOVIA or Inventor.

Special characters in Teamcenter import files

Teamcenter generally excepts all characters that are valid for the underlying database's character set when the server process is configured for a compatible character set.

Teamcenter does not attempt to filter characters (with a few exceptions) when importing files. Teamcenter blocks nonprintable characters for the operating systems character set. However, the nonprintable characters can be imported from a file.

The underlying database rejects characters that do not belong to the database's declared character set. For example, if you set the database to the 8-bit US ASCII (ISO 8859-1) character set, it rejects characters that are not defined for that code page. However, if you set the database for a 7-bit ASCII character set, it allows noncode page 8-bit characters because it does not validate the high bit.

Teamcenter does filter specific strings in situations where their use would raise security concerns due to potential command injection attacks. For example, Teamcenter rejects a semicolon (**;**) or the string **rsh** if they are in a file name.

Additionally, Siemens Industry Software Inc. recommends that you do not use the following characters in item IDs to avoid potential database issues:

- Single quotation mark (**'**)
- Double quotation mark (**"**)
- Slash (**/**)
- Backslash (****)
- Colon (**:**)
- Less than (**<**)
- Greater than (**>**)
- Vertical bar (**|**)
- Tilde (**~**)
- Back tic (**`**)

Exporting variable length arrays

When you export string type variable length arrays (VLA) in a TC XML file, the export process inserts a slash (****) character before the predefined separator character and any slash characters in property values. The predefined separator character is defined in the **GMS_tcxml_string_separator** preference and is a comma (**,**) by default.

For example, when exporting the following VLA string property values:

```
prop[0] “ab,c”
prop[1] “d\ef”
```

The output file contains:

```
prop="ab\,c,d\\ef
```

A slash character is prefixed to the slash and comma (default separator) in the strings. During the import process, the **tcxml_import** utility parses the property correctly and the values are the same at both the exporting and importing site.

Configured BOM export

You can export a configured bill of materials (BOM) using TC XML, providing extensibility and an increase in speed and scalability for this type of export. The export uses the fast traversal method of the bulk loader to quickly stream Teamcenter data to a TC XML file. This functionality is available from the My Teamcenter and Structure Manager applications in the rich client interface.

Teamcenter constructs the BOM for the given root object and configuration rules (revision and variant rules). It traverses the BOM as defined by the closure rules and serializes the traversed BOM data with properties defined by the property set. The inputs required to construct the BOM are passed as a transfer formula in the transfer session options. The transfer formula includes:

- The transfer option set (TOS)

- The transfer mode
- A list of user selected options that override the TOS options
- Session options

You can export both precise and imprecise assemblies with related notes, attachments, absolute occurrences and overrides, alternates and substitutes, and generic design elements (GDE) lines.

Requirement objects

You can transfer requirement objects and their associated data between Teamcenter sites, with or without ownership, and can synchronize replicated data. However, if you require remote check out and check in actions, you must use Multi-Site Collaboration.

You can transfer to other sites (export/import) custom notes along with their Trace link. You can also export notes on Trace links to other sites using PLM XML.

> Note:
>
> PLM XML import of notes on trace links is not supported. Notes on trace links is not supported for briefcase file transfers.

Transferring notes on trace links requires you to use the **REQ_export_notesonlinks** transfer mode.

Vendor management objects

You can transfer vendor management objects between Teamcenter sites, including their roles, such as:

- Supplier
- Manufacturer
- Distributor
- Commercial part
- Manufacturer part

Multifield keys

Multifield keys are identifiers assigned to each object to ensure their uniqueness in the database. Multifield keys definitions are supported for shared objects.

Teamcenter administrators can add multiple properties to define a key. The multifield key is composed of a domain name (the name of the business object type) and a combination of the object's properties. This ensures a unique identifier across all the objects in the domain. Configure multifield keys for objects using the Business Modeler IDE.

Transferring object ownership when sharing data between different schemas

You can share data among Teamcenter sites that have variations in their object schemas by mapping the attributes between objects that differ.

Be aware that ownership of objects can be transferred between Teamcenter sites only when the schemas are identical. To transfer ownership between Teamcenter sites with variant schemas, you must create a custom style sheet (XLST file) that excludes the objects that are not defined at the importing site.

Multiple language support

The Teamcenter multilingual schema contains elements for localizable attribute value representations in one or more languages. This allows you to export and import objects with localizable attributes for display names in more than one language. Teamcenter clients that access the imported data can display the localized attributes in differing languages depending on their locale.

The TC XML file used to transfer objects from and to a multilingual site, contains elements that identify the site primary language, all required languages, and any allowed languages for the site. For exports, unapproved locale attribute values are not included in the TC XML file, and a message is written to the exporter log file stating the attribute value was not exported. For imports, the locale information in the TC XML file and locale information obtained from the importing system is used to determine if localized values in the TC XML file are supported in the importing system. Values for unsupported locales are ignored, and a message is written to the importer log file stating the attribute value was not imported.

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

Consider the following when transferring objects with localized attributes and when monolingual and multilingual sites participate in your data sharing:

- When transferring objects between multilingual sites that have different site primary languages, the import succeeds only when:
  - the importing site supports the site primary language contained in the TC XML file
  - the **TIE_allow_import_with_different_SML** preference at the importing site is set to **true** or **ON**
- Data exchanges are supported between multilingual sites and monolingual sites with the following restrictions:
  - For objects transferred from monolingual to multilingual sites, the site primary language specified in the transfer formula is used to import the localizable attributes. If the transfer formula does not contain the site primary language, the site primary language of the importing (multilingual) site is used to import the localizable attributes. In this case, the locale of the monolingual site must match the site primary language of the multilingual site for the import to succeed.

  - For objects transferred from multilingual to monolingual sites, the locale information in the TC XML file is ignored. Required attribute values are stored in the importing site. Therefore, the site primary language at the multilingual site must be supported at the monolingual site, and all localized attributes must have a site primary language representation at the multilingual site.

- The Teamcenter databases must be character set compatible. For example, transfers between sites using the UTF-8 character set and sites using non-UTF-8 characters (such as Shift-JIS) are not supported.

- You can transfer objects with ownership between multilingual sites that have their required languages common to both sites. The **tcxml_import** and **tcxml_export** utilities support arguments that allow you to specify allowed and required languages.

  Caution:

  When attribute values contain commas in the strings at any of the sites, change the value of **GMS_tcxml_string_separator** at each site to prevent data corruption at the importing site.

- You can synchronize objects between multilingual sites and between monolingual and multilingual sites.

Data sharing configuration

Configuration tasks

Configuring Teamcenter for data sharing involves performing the following tasks:

1. **Define sites**.
2. Configure File Management System (FMS).
3. Create transfer option sets.
4. Set Access Manager (AM) rules to allow transfer of ownership between Teamcenter sites.
5. (Optional) **Create closure rules**.
6. (Optional) Create transfer modes.
7. (Optional) **Configure PDX export**.

Define remote sites

Remote sites are created using the Organization application. You must define sites for all locations that exchange data.

1. Select the top-level sites node  from the **Organization List** tree.

2. In the **Sites** pane, type a descriptive name for the site in the **Site Name** box. For the deployment example, Teamcenter site 1 defines two remote sites: **TcHost1** and the **TcHost2**.

3. Type a unique identifier in the **Site ID** box. For the deployment example, **457655709** and **457141260**.

   > Caution:
   >
   > Each site must be defined at other sites using exactly the same site ID in each definition. For Teamcenter sites, this value is defined during the installation process.

4. Type the URL for your Teamcenter Integration Framework web server in the **TcGS URL** box.

5. Select the **Uses TCXML Payload** check box.

6. Select the **Is HTTP Enabled** check box.

7. Select the **Allow deletion of replicated master objects to this site** check box.

8. Click **Create**.

The site is created and appears in the **Organization List** tree.

Configure attribute mapping for 4GD revisioning

To support revisioning of 4GD objects between versions of Teamcenter, you must configure attribute mapping to support the two methods currently supported for revisioning. You must be a Teamcenter administrator user to configure attribute mapping at both of the sites.

To support Multi-Site and high-level (HL) and low-level (LL) TC XML data sharing (TIE):

1. Attach the **mdl0element_thread_to_wso_thread_mapping.xsl** XSLT file available in the *TC_DATA* directory to both the exporting and importing transfer mode being used for HL TIE. Use the plmxml_tm_edit_xsl command line utility, for example:

   ```
   plmxml_tm_edit_xsl -u=admin-user-name -p=admin-password -g=dba
   -transfermode=TIEImportDefault -xsl_file=%TC_DATA%
   \mdl0element_thread_to_wso_thread_mapping.xsl
   -action=attach
   ```

> Note:
>
> This example uses the TIE default (**TIEImportDefault**) transfer mode. For mapping attributes for Multi-Site transfers, use the **MultiSiteDefaultTM** transfer mode.

2. Ensure the **wso_thread** property of the **Mdl0ConditionalElement** object is set in the property set of transfer mode you use for the data transfer. Include the following clause in the property set:

```
CLASS Mdl0ConditionalElement ATTRIBUTE wso_thread DO
```

After you perform these commands, you can also use the **tcxml_import** utility to transfer data using LL TIE by specifying the XSLT file to use or the option set to use. For example, to specify the XSLT tile:

```
tcxml_import -u=admin-user -p=admin-password -g=dba -xsl=xslt-file-name
 -file=tcxml-import-file-name -low_level
```

To specify the option set tile:

```
tcxml_import -u=admin-user -p=admin-password -g=dba -optionset=option-
set-name
 -file=tcxml-import-file-name -low_level
```

The transfer mode associated with the transfer option has the XSLT file attached in the first step of this procedure.

Creating closure rules

Teamcenter provides standard closure rules that are used by the transfer modes provided. To create additional closure rules, see *PLM XML/TC XML Export Import Administration*.

A closure rule must contain clauses for each type of data a site exports to another site.

You can provide conditional clauses in a closure rule. The closure rule qualifies only if the conditional clause evaluates to **TRUE**.

You can include symbols in a conditional clause. The symbols can be referenced from the transfer option set and can be presented to your users as options they can turn on and off. Enter a symbol in the conditional clause as a string that begins with a **$** character, for example:

```
If $HPGL
```

Closure rules are a component of a transfer mode and therefore must be created before the transfer mode. Closure rules can use various action directives, such as the **SKIP_GRM** action that Teamcenter uses to skip the Generic Relationship Management (GRM) objects for exports. Alternatively, the **SKIP** action skips the secondary objects but includes the primary and GRM objects in the export.

Defining local and remote transfer option sets

A transfer option set (TOS) is defined as a local TOS (for export transfers) or a remote TOS (for import transfers). Data sharing uses this TOS to pull (import) data from the remote system. The TOS has no associated transfer mode, because the transfer mode (TM) and closure rules are defined at the remote site. The remote site TOS is used to populate the export Teamcenter XML (TC XML) file.

Closure rules are filtered based on the options in the remote TOS.

See *PLM XML/TC XML Export Import Administration* to create a transfer option set.

> Note:
>
> A remote TOS is typically imported from a remote site to ensure that it matches the TOS on the remote site. Although you can create a remote site TOS and manually synchronize it to the remote site, Siemens Digital Industries Software recommends that you use **tcxml_import** and **tcxml_export** to provide your remote TOS. Once a TOS is imported, do not change the TOS name or site reference. However, you can add or remove options as long as the TOS at the remote site is updated to reflect the changes you make.

For remote transfer option sets, you define only the TOS name. The name must match the TOS name defined in the remote system. You must define a local transfer option set for exporting data to another site. A local TOS requires that you define an option row in the options table for each symbol that is defined in the associated closure rule clause.

You can determine the options included in a TOS and get a description of their purpose in the PLM XML/TC XML Export Import Administration application.

You can also modify the option settings, remove or add new options, and create new options in this application. The best practice is to create a new TOS and not to modify the Siemens Digital Industries Software supplied transfer option sets. The standard transfer option sets are overwritten when you do a Teamcenter upgrade; any modifications you made are lost.

If you add custom options, you must customize Teamcenter to handle them.

Transfer CATIA alternate shape representations

If you intend to transfer CATIA alternate shape representation objects between Teamcenter sites, you must add the following property set clauses to the **TIEPropertySet** property set at the exporting site:

```
CLASS.POM_alternateSR_userdata:ATTRIBUTE.Shapes_Files:DO
CLASS.POM_alternateSR_userdata:ATTRIBUTE.Shapes_Names:DO
```

These property clause entries export the **Shapes_Name** and **Shapes_Files** attributes for the **POM_alternateSR_userdata** object.

Create version 0 transfer option set

Teamcenter stores two versions of a dataset by default (version 0 and version latest; version 0 is a copy of latest version). When exporting an item or item revision that contains an **IMAN_specification** relation to a dataset, Teamcenter exports two datasets pointing to the same **ImanFile** object. You can change the export behavior to export only version 0 of the dataset with the latest version exported as a stub. To do this, you must add the **opt_from_tc_ds0_only** option to your transfer option set and set its value to **True**.

> Caution:
>
> Do not set this option to **TRUE** when you are transferring objects through a briefcase file. If this option is not set to **FALSE**, the briefcase import fails.

1. In the Teamcenter rich client, open the PLM XML/TC XML Export Import Administration application and select **TransferOptionSet** in the option pane beneath the tree pane.

2. Select the name of the transfer option set you want to use when transferring item or item revisions with related datasets.

3. Click + to add a new option.

4. Select the first column (**Option**) of the new row (at the bottom of the list) and type **opt_from_tc_ds0_only** in the **Option** box.

5. Press the Tab key and type **Zero version export** in the **Display Name** box.

6. Press the Tab key and select **True** from the **Default Value** list.

7. Press the Tab key and type **Export zero version dataset only** in the **Description** box.

8. Press tab and type **Dataset Options** in the **Group Name** box.

9. Click **Create**.

Configuring locally modifiable folders

By default, using Briefcase to export a folder and the objects within it creates replicas of the objects, including the folder, at the target site. Users at the target site cannot make changes to the folder, for example, by adding local objects to the folder.

In certain cases, you may want the users at the target site to be able to update the contents of their local version of the folder. Doing so allows them to add objects to the folder they can then export to the owning site.

To allow folders to be updated locally at a target site, add the **opt_keep_folder_local** option to the transfer option set (TOS) used for export and to the TOS used for import at the target site. This option is used only with low-level unconfigured transfer option sets.

When a folder is exported and imported with these options enabled, the imported folder is created locally at the target site with its child objects created as replicas. If the owning user does not exist at the target site, the importing user becomes the owner and can edit folders. Other users at the target site can also edit the folder if they are in same group as the owning user.

Exporting folders

When exporting the folder and its contents, check **Export folder as local** in the **TOS General Options** choices. Explicitly select the folder for export if root objects need to be inserted in the folder on import.

Importing folders

When importing, check **Export folder as local** in the **TOS General Options** choices. If more than one folder is selected for import, root objects are inserted into the first folder.

Local objects persist if a folder is later re-imported with additional updates from the owning site.

Exporting and importing from the command line

You can export and import locally modifiable folders from the command line using **tcxml_export** and **tc_xml_import** (respectively) with the **-session_options** including **opt_keep_folder_local:true**.

The export command has the following form:

```
tcxml_export -u=<user> -p=<pwd> -file=<file_name> -folder=exportObjects
  -optionset=TIEUnconfiguredLLBCZExportDefault -briefcase -targetsites=<target_site>
  -session_options=opt_keep_folder_local:true
```

The import command has the following form:

```
tcxml_import -u=<user> -p=<pwd> -briefcase -file=<file_name>
  -session_options=opt_keep_folder_local:true
```

When the **-folder** option is included, root objects in the Briefcase are imported in the specified folder.

Set a rule to allow transfer of ownership

By default, the Access Manager (AM) rules do not allow transfer of ownership from one Teamcenter site to another. You must change the default value to enable this capability.

1. In Access Manager, select **Has Class (POM_application_object)→Import/Export** in the rule tree.

2. Click ✔ to grant **Transfer Out** () privileges.

3. Click **Modify** and click  .

2. Exchanging data using Briefcase files

Sharing data using Briefcase files

You can use Briefcase files to share Teamcenter data with sites not connected using Multi-Site Collaboration or Teamcenter Integration Framework. You can share data with other Teamcenter sites or with non-Teamcenter (unmanaged) sites. Unmanaged sites can use Briefcase Browser to work with Teamcenter Briefcase files.

Prerequisites

One or more licenses are required to work with Briefcase files.

Importing and exporting Briefcase files

Each Teamcenter site that is importing or exporting Teamcenter data must have a Briefcase license.

Using optional Briefcase features

Sites using the following optional import and export features must each have an Advanced Multi-Schema Exchanger license.

- Briefcase export and import dry run and validation capabilities
- Briefcase preview and compare capabilities
- Advanced Multi-Schema Exchanger

Contact your Siemens Digital Industries Software customer service representative for more information on Briefcase licenses.

Data sharing with the rich client

Share data using the rich client's **Import** and **Export** choices on the **Tools** menu.

See **Exchanging data using Briefcase files** for more information on using the rich client to exchange Teamcenter data.

Command line data sharing

Teamcenter provides several command line utilities to share data with other sites or third-party products, such as:

plmxml_export

Exports objects from Teamcenter in PLM XML format.

plmxml_import

Imports objects to Teamcenter from a specified PLM XML file.

tcxml_export

Exports objects from Teamcenter in TC XML format. The output optionally be formatted as a Briefcase file.

tcxml_import

Imports objects into Teamcenter from a TC XML file.

pdx_export

Exports data from Teamcenter in PDX format.

In a standard installation, these utilities are available without any changes required by the system administrator. However, to use any of the utilities to exchange **ImanFile** objects, the Teamcenter distribution image must be installed locally or the system path must include the path to the FSC library. For Windows systems, add **%TC_ROOT%\fsc\lib**. For Linux systems, add **$TC_ROOT/fsc/lib**.

Incremental change export

You can export incremental change information in a briefcase file. The incremental change data export is based on object timestamps. The importing site determines objects are successfully imported and reports the status to the exporting site which then uses the information to update the import/export record (IXR) of the objects at the exporting site. Therefore, exporting an incremental change briefcase file is a two-step process.

For the import to succeed, the system administration objects, such as effectivity and revision rules, used to configure the exported structure must exist at the importing site. The import process searches for the system objects at the local site by name, the contents of the objects is not verified. If a local system object is not found the import process logs an error in the import log file.

During the export process, if a revision rule results in an imprecise line, the exporter includes the latest revision. Unconfigured lines and any unchanged objects configured by incremental change that have already been exported are exported as stubs (**BOMLine** objects as full objects with **PSOccurrence** objects as stubs). Additionally, new BOM lines added to a structure that are not tracked by incremental change are exported as stubs and all substructure to the BOM lines are ignored.

Any BOM line not tracked by incremental change that is deleted from an exported structure is not included in the exported briefcase file. The importer deletes (inferred delete) the object at the importing site when the structure exists at the importing site.

You can export a briefcase file with multiple root objects that are configured by a single revision rule or with each having a different revision rule (collaboration context). A structure context is a configurable structure that consists of one or more root objects sharing the same configuration. A collaboration context is a container object containing one or more structure contexts, each of can have a different configuration. The configuration of the structure context is defined by a configuration context, and may include revision rules, variant rules and closure rules.

Closure rules are also system administration objects. The closure rules for a collaboration context must exist at the importing site or be imported separately prior to importing a briefcase file containing the collaboration context objects.

Collaboration and structure objects are workspace objects. They cannot be revised, but they can be managed with incremental changes. The change object affects the structures contained in the collaboration context object. However, you cannot export incremental change data affecting objects specific to the manufacturing data model in a briefcase file.

Exchanging Briefcase files

You can use Briefcase files to exchange data with other Teamcenter sites that do not participate in a data exchange environment or for sites that are temporarily offline.

You can also use Briefcase files to exchange data with sites that do not manage their data with Teamcenter (*unmanaged* sites) such as supplier sites. Unmanaged sites must have Briefcase Browser installed and configured to work with Briefcase files. Subsystem designers and manufacturers, component suppliers, and build-to-specification suppliers are examples of suppliers that may have unmanaged sites.

Using Briefcase files to exchange data:

- Allows variability of data models.
- Supports metadata, JT, NX, and CATIA data.
- Requires data push (manual file transfer, such as email attachment) unless you are using the Supplier Collaboration solution.
- Supports partial BOM export.
- Supports delta export from a full assembly and from subassemblies.
- Supports round-trip exchange of CAD data (stubs for remote objects not checked out to site in the Briefcase file).
- Supports configured export by revision rules and variant rules.
- Provides Briefcase package preview and compare in the Teamcenter rich client.
- Supports export of CAD incomplete assemblies.

If your site is a *managed* Teamcenter site, you must perform the migration process before using Briefcase Browser at your site.

Caution:

Briefcase Browser does not support the **Item:sci0IsIMDSObject** property added by the International Materials Data System (IMDS) template. Therefore, if you are exchanging data with an unmanaged site, you cannot have this template installed in Teamcenter. If you do have it installed, you must manually remove the property occurrences from the Briefcase file.

Briefcase Browser supports both NX and CATIA CAD data. You can have only one plug-in installed at a time. However, there are some differences between the plug-ins for the CAD formats.

- For CATIA CAD data, drawing files in Briefcase Browser are confined to the data model where the drawing is related to an item revision of the same item.
- The CATIA schema is validated strictly by the rules of the supplied schema.
- When you open a partial briefcase file in Briefcase Browser, there may be differences in the structure view from what is displayed in Teamcenter, for example, Briefcase Browser shows unselected NX siblings as stubs, however the CATIA plug-in ignores them.
- For CATIA data you can export a partial BOM export and assemblies with variant rules.

Using NX data with Briefcase

For Briefcase packages containing NX data, infer delete (deletion if not locally owned) is allowed for occurrences in any type of export in the following cases.

- when a supplier owns the object locally
- when the object is checked out to the supplier site

This includes exports with incomplete, variant, or partial CAD data. For Briefcase packages with CATIA data, infer delete of occurrences is not allowed for exports with incomplete, variant, or partial CAD data.

Exports to unmanaged sites do not create an export record (IXR) in the database. For performance reasons, the Briefcase accountability table is used instead for unmanaged exports. Therefore, the object is displayed in Teamcenter with no value in the **Exported To** column in the rich client.

You must set the **include_all_objects** option in the **CustomMappings.xml** file to **true** if you are adding objects to a CAD part in the CAD application. Otherwise, the new objects are not shown when you import the Briefcase file into Teamcenter.

Unmanaged sites exchanging NX data must supply English characters for the item ID, revision, and name attribute values. Other locales are not supported in this case.

NX alternate representations

Briefcase Browser cannot operate on NX alternate representations (**UGALTREP** objects). For performance reasons, this clause is not included in the unmanaged transfer option set. If you want this data to be displayed in Briefcase Browser, add the following clause to the **TIEUnmanagedExportforNX** transfer option set:

```
CLASS.ItemRevision:TYPE.UGALTREP:RELATIONP2S.IMAN_UG_altrep:Process+traverse:
$opt_nx_altrep=="true"
```

Parsing NX clone output log files

By default, when you import a Briefcase file created from an NX assembly or part, Teamcenter does not parse the clone output log file (*ug_clone_import_console_output.log* for the **ug_clone** utility or *tcin_import_import_console_output.clone* for the **tcin_import** utility). If you require parsing of the clone output log file for debugging, you must set the **BC_NX_DEBUG** environment variable value to **TRUE** at the target site. However, do this only if absolutely necessary as this may cause errors during the import.

Supplier site exchange

For security or logistics reasons, you may need to exchange information with a supplier that does not have an online connection to any of your Teamcenter sites. Using Briefcase allows you to provide the shared data in an archive file that can be physically moved or electronically transferred to or from the supplier site.

You can also check out objects to a site. This functionality creates a briefcase archive that has the checked-out object, the associated objects required to properly lock the object for modification, and the site reservation object that contains a remote site attribute. If the remote site attribute matches the importing site, Teamcenter creates a reservation object associated to the object at the importing site. This allows users at the importing site to locally check out and modify the objects (provided they have the proper privileges) even though the objects are replicas and not owned by the site. An object that is checked out to another site has its **Checked Out by property** set to the remote site. The following objects can be checked out to a site:

- **BOMView**
- **BOMViewRevision**
- **Dataset**
- **Form**
- **Item**
- **ItemRevision**

Tip:

The view type data is not exported with the **PSBOMView** data in TC XML. The import process uses the default view type when importing **PSBOMView** data.

Therefore, set the **PSE_default_view_type** preference at the target site to the view type value of the assembly at the source site. Usually, this is the same as the source sites default view type. Do so when transferring **PSBOMView** data using a briefcase file, **tcxml_export** and **tcxml_import**, or any utility that uses TC XML metadata.

This import behavior is designed to be consistent with Structure Manager.

The following table identifies the related objects that are checked out to a site:

Object selected for checkout	Associated objects checked out
Item	Item primary (master) form
ItemRevision	Item revision primary form
Dataset	Named references that are work space objects

The checkout to site functionality also supports tiered suppliers. A supplier that is a hub site (tier-1 supplier) can use the **Check-Out to site** command to check out objects to one of their suppliers (tier-2 supplier) for modifications. The owning site attribute is set to the exporting site (tier-1 supplier), allows the importing site to modify the data and send it back to the hub site.

Note:

If a tier-2 supplier does not use the **Check-In from site** command before sending an object back to the tier-1 supplier, the tier-1 supplier must perform the **Check-In from site** command twice on the object to release the lock.

A site that has checked out an object to a site can cancel the site checkout to restore the reservation object to its precheckout state. Datasets are not restored to their precheckout state but the site checkout lock is removed.

Remote checkout and the **Check-Out to site** command are mutually exclusive and Teamcenter disables the commands to prevent the other action.

Configuring Briefcase data sharing

Configure Briefcase for site checkin and checkout

You must assign certain privileges to use Briefcase with the **Tools→Check In/Out→Check Out To Site** commands in a non-SSO configuration. Give the following access privileges to the Teamcenter user specified in the Teamcenter Integration Framework configuration.

Symbol	Privilege
	**Read**
	**Write**
	**Export**
	**Import**
	**Transfer out**
	**Transfer in**

If these access privileges are not set, the transfer back to Teamcenter fails because the Teamcenter user does not have the required object access privilege.

1. Log on to Teamcenter as a user with administrator privileges.
2. Choose **Edit→Options** and click **Filter**.
3. Click the **Create a new preference definition** button () and type **GMS_tcxml_string_separator** in the **Name** box.
4. Select **GMS Offline.Import** from the **Category** list and **Site** from the **Protection Scope** list.
5. (Optional) Type **MP2 Briefcase string separator** in the **Description** box.
6. Type a comma (**,**) and a space in the **Values** box.
7. Click **Save**.

To export an assembly that is either:

- Cloned from a replica or an assembly that is checked out to a remote site
- Created using the **Save As** command on a replica or an assembly that is checked out to a remote site

You must set the **opt_based_on** option to **FALSE** in the transfer option set or uncheck **Include IMAN_based_on** in the **Export to Briefcase** dialog box.

Site checkin after import

The **GMS_site_checkin_after_import** preference controls whether or not a Briefcase file that contains objects checked out to a site are automatically checked in when it is imported. If set to **true**, the objects are checked in on import.

Object-based site checkout

To enable object-based site checkout of objects in exported Briefcase files, set the **Object_based_site_check_out** preference to **true**.

When object-based site checkout is enabled and you select an item revision as the root object, the selected revision, its helper objects, and the related item and its helper objects are candidates for site checkout. If an item is selected for site checkout, based on the revision selector defined in **TIEUnconfiguredExportDefault**, the corresponding item revision and its helper objects are candidates for site checkout.

By default, Teamcenter uses the **TIEUnconfiguredExportDefault** option set for traversal. You can use a different option set by creating the **option_set_for_site_check_out** preference and setting its value to the desired option set.

This same behavior applies to **Cancel check out** and **Site check in**.

Configure Teamcenter for Briefcase mail notification

Configure Teamcenter to create an envelope object containing your Briefcase package in the mailbox folder automatically when you create a Briefcase package by setting the **Briefcase_tcmail_notification** preference to a value of **TRUE**. Set the preference value to **FALSE** to disable envelop object creation.

Configure the Briefcase Viewer

Perform the following steps before using the Briefcase Viewer for the first time or when you need to change the configuration for the package content.

1. Choose **Window→Preferences** to display the **Preferences** dialog box. Ensure **Briefcase Viewer** is selected.

   (When later updating the Briefcase Viewer configuration, you can also click **Comparison Preferences**  in the **Briefcase Preview** pane to open the **Preferences** dialog box.)

2. In the **Preferences** dialog box, click **Explorer**. Make note of the location of the default configuration folder. This is the location that contains the configuration files required by the Briefcase Viewer.

3. Browse to the **site configuration files** directory (containing the TC XML schema file, **custommappings.xml** file, and so on).

4. Copy the configuration files to the default directory location.

Customize Briefcase Viewer icons

You can customize the icons used to represent different object types in the **Briefcase Preview** pane. Custom icons are mapped to objects with various ownership types in the *custom-icons.properties* file. Using customized icons has the following requirements:

- Icons must be 16 x 16 pixel bitmap images or SVG images. Most bitmap formats are supported, including *.gif*, *.png*, and *.jpg*.
- Icon images must be stored in an *\icons* directory in **your Briefcase Viewer configuration folder**.
- Create a *custom-icons.properties* ASCII file defining custom icon mappings and store the file in your Briefcase Viewer configuration directory.

Mapping custom icons

Custom icons appear in place of the default icons. Create an ASCII *custom-icons.properties* file to specify which custom icons override which specific default icons. Save this file in your Briefcase Viewer configuration folder.

Each line in the file defines how a particular image file overrides one or more default icons. If *custom-icons.properties* contains more than one line, icon mappings are applied in the order of the most-specific context to the least-specific context.

Each line in *custom-icons.properties* has the following form:

ownership, *object_type*, *file_name*

Valid values are as follows:

ownership

The ownership of the related object type. The custom icon is used as follows:

all	All owners.
local	The object is owned locally.
stub	Stub objects.
replica	Objects that are replicas.
checked out	Objects that are checked out.
ownership transfer	Objects that are marked as having ownership transferring to another site.

object_type

The Teamcenter object type. Specific values are defined in your data model using BMIDE. A value of **all** specifies the custom icon will be used for all object types.

file_name

The name of the image to use as the icon. The image must be stored in an *\icons* directory in your Briefcase Viewer configuration folder.

Lines starting with a **#** symbol are considered as comments and are ignored.

Save the updated *custom-icons.properties* file and reload a Briefcase file in the Briefcase Preview pane to display the custom icons.

Examples

Examples of overriding default icons with custom icons:

- Assign the icon *stub_generic.png* to all stub objects.

```
stub, all, stub_generic.png
```

- Assign the icons *stub_docrev.png* to document revision stubs, *stub_itemrev.png* to item revision stubs, and *stub_generic.png* to all other stub objects.

```
stub,all,stub_generic.png
stub,documentrevision,stub_docrev.png
stub,itemrevision,stub_itemrev.png
```

- Assign the icons *stub_docrev.png* to document revision stubs, *replica_docrev.png* to document revision replicas and *docrev.png* to all other document revision object types.

```
stub,documentrevision,stub_docrev.png
replica,documentrevision,replica_docrev.png
all,documentationrevision,docrev.png
```

Allocate memory for comparing and previewing large Briefcase files

When using the Rich Client to preview or compare large Briefcase files (containing 100,000 or more objects), you may need to increase the memory allocated for the Rich Client as follows:

1. Close any running instances of the Rich Client.

2. In an ASCII editor, open the file *TC_ROOT\portal\portal.bat*.

3. Change the value specified in the `set VM_XMX` line as follows:

```
set VM_XMX=2048m
```

4. Restart the Rich Client.

When previewing or comparing Briefcase files containing significantly more than 100,000 objects, you may need to increase the value of **VM_XMX** higher than 2048m.

Configure exchange of 4th Generation Design data

4th Generation Design (4GD) data can be exported and imported only using TC XML Briefcase files. To do so, set the **GMS_offline_use_TcGS** preference to **FALSE**.

1. In the rich client, choose **Edit→Options**.

2. In the **Options** dialog box, click **Filters** and type **GMS_offline_use_TcGS** in the **Filters** box.

3. Click **Edit**, type **false** in the **Value** box, and click **Save**.

4. To change the default value (**Latest Working**) for the revision rule used to configure a partition:

    a. Type **TC_config_rule_name** in the **Filter** box.

    b. Click **Edit**, type the revision rule name in the **Value** box, and click **Save**.

    You can override the default revision rule value by selecting the desired rule name from the **Revision Rule** list.

Working with unmanaged sites

You can import and export configured Briefcase files to and from unmanaged sites in the My Teamcenter and Structure Manager rich client applications. You must have Teamcenter set up with unmanaged sites and must use Briefcase Browser at those sites to create and modify Briefcase files.

Configure Teamcenter for Briefcase Browser

A Briefcase file that is exported to an unmanaged site is a configured export that uses the **TIEConfiguredExportDefault** transfer option set.

The unmanaged site must be defined for each unmanaged site in the Organization application. The site must be defined with the **Is Unmanaged**, **Is Offline**, and **Uses TCXML Payload** options selected.

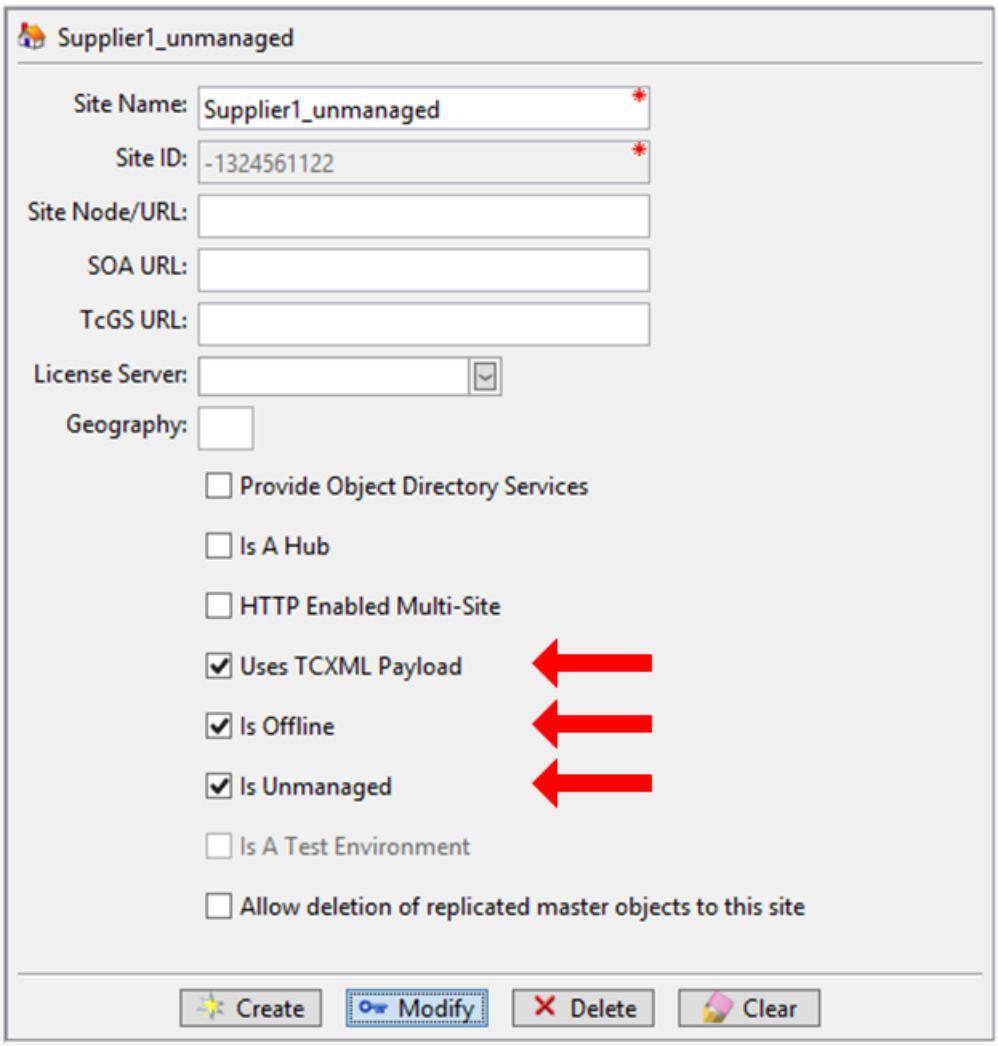
1. In your Teamcenter command shell, import the required transfer modes as follows:

```
tcxml_import -u=tc-admin-user -p=password -file=%TC_DATA%\defaultTcxmlScopeRules.xml
-scope_rules -scope_rules_mode=overwrite

tcxml_import -u=tc-admin-user -p=password
-file=%TC_DATA%\siteConsolidationInternalClosureRules.xml
-scope_rules -scope_rules_mode=overwrite
```

2. Log on to Teamcenter as a user with Teamcenter administration privileges and open the Organization application.

3. Define a site for the unmanaged site. Select the **Uses TCXML Payload**, **Is Offline**, and **Is Unmanaged** options.



4. Ensure the following preference values are set at the site location:

   TC_gms_export_default_transfermode=TIEExportDefaultTM

   TC_gms_import_default_transfermode= TIEImportDefault

   TC_show_checkedout_icon = TRUE

   The following preferences are optional. Set these values as required.

   BRIEFCASE_export_CAD_incomplete = FALSE

You can set this to **TRUE** if you want to allow incomplete CAD exports. Siemens Digital Industries Software recommends the following settings:

- **Protection Scope**

  User

- **Category**

  General

- **Environment**

  Disabled

- **Multiple**

  Single

- **Value**

  FALSE

BRIEFCASE_import_validation_rule_item = *item_id/rev_id*

item_id/rev_id indicates the item that contains the dataset validation rules. For example, set the value to **000123/B** for the dataset attached to item (**item_id**) **000123** revision (**rev_id**) **B** as the validation rule dataset.

Siemens Digital Industries Software recommends the following settings:

- **Protection Scope**

  User

- **Category**

  General

- **Environment**

  Disabled

- **Multiple**

  Single

Briefcase Browser site configuration files

Briefcase Browser reads the XML configuration files in the **bbworkspace\configurations** directory. Each configuration has a separate directory that contains the **site-id.properties** file as a minimum.

The **create_briefcase_browser_config_package** utility creates a compressed package file of the configuration files that you can provide to your suppliers. The utility takes the path to the Briefcase Browser installation ZIP file (**-bbDir** argument) and generates a managed (OEM) site ZIP file containing the OEM configuration package.

```
create_briefcase_browser_config_package -u=username -p=passwd -g=dba
-bbDir=C:\apps\bb
```

The package name is the OEM site name. It contains the **CustomMappings.xml** file with the OEM site ID set as the target site ID. It also contains the **TCXML.xsd** schema file, generated from the OEM and the OEM configuration file.

site-id.properties

Defines the unmanaged site ID. This value can be any numeric sequence not starting with a zero and is not required to meet Teamcenter site ID requirements. If this file is not present, Briefcase Browser does not allow you to create a Briefcase file. You can create or modify this file manually or use the **Preference** dialog box in Briefcase Browser.

It must also contain the following additional configuration files:

TCXML.xsd

Defines the schema that Briefcase Browser uses to load, display, and create Briefcase files. This file is typically supplied by the managed (OEM) site to the supplier (unmanaged) site. It is available in the *TC_DATA* directory of the Teamcenter installation.

If a **TCXML.xsd** file does not exist in the configuration directory for your site, Briefcase Browser uses the standard Teamcenter schema. **TCXML.xsd** is required in this directory for comparing and previewing Briefcase files.

> Note:
>
> The uniform resources identifiers (URIs) that appear in the headers of PLM XML and TC XML files serve as namespace names, are unique identifiers for an XML vocabulary. Although they are URIs, they are not used to identify and retrieve web addresses.

CustomMappings.xml

Defines the following information about the Teamcenter (OEM) site with which you exchange Briefcase files:

- The site name.

- The site ID.
- The Teamcenter version.
- If your site must use the **auto_baseline** (baseline) feature when exchanging updated Briefcase data.
- If the site allows precise or imprecise structures.
- If your site must create all replica objects as stubs (**include_all_objects=false**). When writing reference objects as stubs, Briefcase Browser does not package any corresponding CAD data in the Briefcase package. If this value is set to **true**, Briefcase Browser writes all objects as full objects in the Briefcase file reference replica parts.
- If your site must include full objects for all checked-out parts and locally owned parts (**include_modified_objects_only=false**). If this value is set to **true**, Briefcase Browser writes local parts as full objects. Parts checked out to your site as full objects are also written as full objects if they are modified since the file was last saved, including their parent and grandparent objects. If objects checked out to your site have not been modified, they are written as stub objects and the corresponding part files are not included in the Briefcase package.
- The separator character the site requires between the item ID and revision.
- Custom type mapping to OEM objects.
- Release status values the managed (OEM) site allows you to assign to parts or assemblies that you create in the **release_status_name** elements. Briefcase Browser displays the values that you supply in **release_status_name** elements in the **Release Status Name** list in the **Preferences** dialog box. The value you select in the **Preferences** dialog box is assigned as the release status of any new CAD parts or assemblies you create. The value must match one of the valid values for the Teamcenter release status at the OEM site.
- Custom forms related to item revisions to process when exchanging data, such as business object forms with mass-related properties. When **ir_to_form** is set to **true**, the listed forms are processed. If **ir_to_form** is set to **false**, the forms are ignored. See the **ir_to_form** section in the following sample for examples.

Sample content for this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<custom_mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:noNamespaceSchemaLocation="../bce-core/CustomMappings.xsd">
   <oem name="your_oem_name" site_id="87654321" tc_version="Target-TC-Version"
        auto_baseline="false" generate_validation_xml="false" is_precise="true"
        separator="/" object_name_separator="/" bomview_type_name="view"
        include_modified_objects_only="false" include_all_objects="false"
        extract_to_sub_directory="false" process_reference_briefcases="true">
     <release_status_name>Snapshot</release_status_name>
     <release_status_name>TCM Released</release_status_name>
```

```
   <release_status_name>R2InWork</release_status_name>
</oem>

<ir_to_form enabled="false">
  <form_type>custom_form_type</form_type>
  <relation>custom_relation</relation>
</ir_to_form>

<item_type_mapping custom_type="Item">
  <multi_field_key>
    <attribute>item_id</attribute>
  </multi_field_key>
  <helper_object_map custom_type="ItemRevision"
          ootb_type="ItemRevision"/>
  <helper_object_map custom_type="Item--Master"
          ootb_type="Item--Master"/>
  <helper_object_map custom_type="ItemRevision--Master"
          ootb_type="ItemRevision--Master"/>
</item_type_mapping>

<item_type_mapping custom_type="CAD">
  <multi_field_key>
    <attribute>item_id</attribute>
  </multi_field_key>
  <helper_object_map custom_type="CAD--Revision"
                     ootb_type="ItemRevision"/>
  <helper_object_map custom_type="CAD--Master"
                     ootb_type="Item--Master"/>
  <helper_object_map custom_type="CAD--Revision--Master"
                     ootb_type="ItemRevision--Master"/>
</item_type_mapping>

<item_type_mapping custom_type="B4Item">
<multi_field_key>
  <attribute>item_id</attribute>
</multi_field_key>
<helper_object_map custom_type="B4ItemRevision"
                   ootb_type="ItemRevision"/>
<helper_object_map custom_type="B4ItemMaster"
                   ootb_type="Item--Master"/>
<helper_object_map custom_type="B4ItemRevisionMaster"
                   ootb_type="ItemRevision--Master"/>
/item_type_mapping>

<item_type_mapping custom_type="CommercialPart">
  <multi_field_key>
    <attribute>item_id</attribute>
  </multi_field_key>
  <helper_object_map custom_type="CommercialPart--Revision"
                     ootb_type="ItemRevision"/>
  <helper_object_map custom_type="CommercialPart--Master"
                     ootb_type="Item--Master"/>
  <helper_object_map custom_type="CommercialPart--Revision--Master"
                     ootb_type="ItemRevision--Master"/>
</item_type_mapping>

<item_type_mapping custom_type="Design">
  <multi_field_key>
    <attribute>item_id</attribute>
  </multi_field_key>
```

```
    <helper_object_map custom_type="Design--Revision"
                       ootb_type="ItemRevision"/>
    <helper_object_map custom_type="Design--Master"
                       ootb_type="Item--Master"/>
    <helper_object_map custom_type="Design--Revision--Master"
                       ootb_type="ItemRevision--Master"/>
  </item_type_mapping>

  <item_type_mapping custom_type="Part">
    <multi_field_key>
      <attribute>item_id</attribute>
    </multi_field_key>
    <helper_object_map custom_type="Part--Revision"
                       ootb_type="ItemRevision"/>
    <helper_object_map custom_type="Part--Master"
                       ootb_type="Item--Master"/>
    <helper_object_map custom_type="Part--Revision--Master"
                       ootb_type="ItemRevision--Master"/>
  </item_type_mapping>
</custom_mappings>
```

The **auto_baseline** attribute indicates whether or not your Briefcase Browser provides autobaseline functionality. If this is set to **true**, the baseline feature is enabled. The template file has this value set to **true**. If it is set to **false**, Briefcase Browser does not automatically revision objects. If the attribute is not defined in this file, Briefcase Browser performs automatic revisions as if the attribute was set to **true**.

If the Briefcase file you are exchanging must contain precise assembly structures, set the **is_precise** attribute to **true**. Set this value to **false** when imprecise assembly structures are allowed.

If you intend to add objects to CAD parts in the CAD application, set the **include_all_objects** attribute to **true** to include them when you save the Briefcase file for exporting back to Teamcenter.

You can have any number of release status elements. These become the selection list for **Release Status** in your Briefcase Browser **Preferences** dialog box.

You map custom CAD parts types to OEM types by assigning a custom **Item** type to the items in the TC XML data created for your CAD part. You define the subtype of the **Item** type as a helper object, for example:

```
<helper_object_map custom_type="CADPart" ootb_type="Item"/>
```

This causes the TC XML data generated for custom **CADPart** supplier owned parts to be created as **Item** objects at the OEM site.

> Note:
>
> The custom item type must be defined in the schema (**TCXML.xsd**) file used by Briefcase Browser.

You can have multiple custom types defined in this file. Briefcase Browser locates the matching custom type in the file and maps the file to the defined OEM object. If a custom type part is not found in the **CustomMappings.xml** file, the first custom type defined is used. If there are no custom types defined, Briefcase Browser maps the part to the standard Teamcenter type.

CustomDatasetMappings.xml

This XML file contains valid dataset extensions and relation types supported for Non-CAD files. Each dataset extension specifies whether it is a primary (master), dataset name, **ref_names**, and relation to be used while generating TC XML. Each relation type specifies the display relation for the **Add Dataset** dialog box and the actual relation to be used for generating TC XML. If this file is not found, an OOTB file is used. You can add additional non-CAD types to this file.

> Caution:
>
> If your system has CAD dataset types other than the primary dataset, and the dataset type follows the model file naming convention, you must add an entry representing the dataset. Add the entry with the **is_master** attribute set to **False**, for example:
>
> ```
> <cad_data_set_mapping is_master="False" dataset_type="UGPART"
> ref_names="UGPART" relation=""/>
> ```
>
> This prevents an error from occurring when you open or synchronize an updated CAD assembly in Briefcase Browser.

```
<?xml version="1.0" encoding="utf-8"?>

<!-- XML file containing valid dataset extensions and relation types supported for
Non-CAD files.
  Each dataset extension specifies if it is a master, dataset name, ref_names and
relation to be
  used while generating TCXML. Each relation type specifies the display relation for
the Add Dataset
  dialog and the actual relation to be used for generating
  TCXML. -->
<custom_data_set_mapping xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<data_set_mapping extension="prt">
<cad_data_set_mapping is_master="True" dataset_type="UGMASTER" ref_names="UGPART"
relation=""/>
<cad_data_set_mapping dataset_type="UGPART" ref_names="UGPART" relation=""/>

</data_set_mapping>
        <data_set_mapping extension="doc">
        <cad_data_set_mapping dataset_type="MSWord" ref_names="word"
        relation="IMAN_reference"/>
</data_set_mapping>
        <data_set_mapping extension="docx">
        <cad_data_set_mapping dataset_type="MSWord" ref_names="word"
        relation=""/>
</data_set_mapping>
        <data_set_mapping extension="xls">
        <cad_data_set_mapping dataset_type="MSExcel" ref_names="excel"
        relation=""/>
</data_set_mapping>
```

```
        <data_set_mapping extension="jpg">
        <cad_data_set_mapping dataset_type="JPEG" ref_names="JPEG_Reference"
        relation=""/>
</data_set_mapping>
        <data_set_mapping extension="jt">
        <cad_data_set_mapping dataset_type="DirectModel" ref_names="JTPART"
relation="IMAN_Rendering"/>
</data_set_mapping>
            <relation_type actual_relation="IMAN_specification"
display_relation="specification"/>
            <relation_type actual_relation="IMAN_Rendering"
display_relation="rendering"/>
            <relation_type actual_relation="IMAN_manifestation"
display_relation="manifestation"/>
            <relation_type actual_relation="IMAN_reference"
display_relation="reference"/>
</custom_data_set_mapping>
```

cad_to_tc_attribute_map.xml

Defines the mapping of user-defined NX attributes to qualified Teamcenter object attributes. The user-defined attributes must be part of the configured TC XML schema (**TCXML.xsd** file). This allows you to define attributes for qualified objects in addition to the required attributes.

> Caution:
>
> You can map only one CAD part attribute to one Teamcenter attribute (1-to-1 map). Mapping a CAD part attribute to multiple Teamcenter attributes or a single attribute to multiple item types can corrupt or lost data.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Only one CAD2TC_attribute_mappings that can contain more than one
     attribute map. These mappings will hold a list of CAD attributes
     that map to teamcenter attribute -->

<CAD2TC_attribute_mappings xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="cad_to_tc_attribute_map.xsd">

  <!-- cad_part_attr should be unique for each attribute_mapping -->
  <attribute_mapping cad_part_attr="PART_MATERIAL" tc_attr="nisPartMaterial"
                     tc_type="CAD--Revision--Master"/>
  <attribute_mapping cad_part_attr="PART_THICKNESS" tc_attr="nisPartThickness"
                     tc_type="CAD--Revision--Master"/>
  <attribute_mapping cad_part_attr="CALC_WEIGHT" tc_attr="nisCalculationWeight"
                     tc_type="CAD--Revision--Master"/>
  <attribute_mapping cad_part_attr="CALCULATIVE_COEFFICIENT"
                     tc_attr="nisCalculativeCoefficient"
                     tc_type="CAD--Revision--Master"/>
  <attribute_mapping cad_part_attr="Z_CENTROID" tc_attr="nisGravityZ"
                     tc_type="ItemRevision--Master"/>
  <attribute_mapping cad_part_attr="Y_CENTROID" tc_attr="nisGravityY"
                     tc_type="CAD--Revision--Master"/>
  <attribute_mapping cad_part_attr="X_CENTROID" tc_attr="nisGravityX"
```

```
                    tc_type="CAD--Revision--Master"/>
    </CAD2TC_attribute_mappings>
```

visible-attributes.xml

Defines the attributes that are displayed in Briefcase Browser properties views. By default, Briefcase Browser displays all qualified attributes. If this file is present, Briefcase Browser limits the attributes in the properties views to the attributes defined in this file. The following is sample content for this file:

```
<visible_attributes>

    <group name="Item">
        <attribute ootb_type="Item" name="item_id"/>
        <attribute ootb_type="ItemRevision" name="object_name"/>
        <attribute ootb_type="ItemRevision"
name="object_description"/>
    </group>

    <group name="Admin">
        <attribute ootb_type="POM_imc" name="POM_imc"/>
        <attribute ootb_type="Group" name="Group"/>
        <attribute ootb_type="User" name="User"/>
    </group>

</visible_attributes>
```

attributes_text_locale.xml

Defines the localized values for the language indicated by the name of the directory containing the file. The directory name consists of a two-character locale and two-character country code such as **en_US**. The language directories must be in the **lang** directory in the site's configuration directory.

For example, the following shows the US English (**en_US**) and the Japanese localization (**ja_JP**) directories:

Briefcase Browser reads the TC XML file content and displays the object property names as defined in the file. The properties are database field names that may not be readable by your users. You can use this file to map the database field names to usable localized names.

Your **attributes_text_locale.xml** files must be UTF-8 encoded. They must also contain the following element as the root XML element in the file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

If the property names are not displayed in your locale language, use a different text editor for UTF-8 encoding.

Following is sample content for this file in the Japanese locale:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<textsrv filename="attributes_text_locale.xml">

<key id="inbox_type">志信トレイ</key>
<key id="inbox_name_1">実行タスク</key>
<key id="inbox_name_2">追跡タスク</key>
<key id="type_label">タイプ</key>
<key id="object_label">オブジェクト</key>
<key id="property_label">プロパティ</key>
<key id="msg_string">フォルダ倉も痰存在癌宝癌ん。</key>

<!--generic_shell.cxx-->
<key id="shell_string">ObjectListBox Shell</key>

<!--objectlistbox.cxx-->
<key id="ownerid_string">所有者ID</key>
<key id="date_string">作房日</key>
<key id="cannot_string">読込盈丿帳</key>
<key id="readOnlyMsg">読込盈専用</key>
<key id="app_encapsulation_label">アプリケーション覿約</key>
<!--tool.cxx-->

<!--distributionlist.cxx  envelope.cxx-->
<key id="mail_label">メール</key>

<!--form.cxx-->
<key id="form_def_label">フォーム定義ファイル忿</key>
<key id="structure_group_label">構房グループ</key>

<!--bomview.cxx-->
<key id="bomview_label">BOMビュー</key>
<key id="product_label">製弩</key>
<key id="PSM_label">PSM</key>
<key id="item_id_string">アイテムID</key>
<key id="dataset_id">データセットID</key>
<!--! <MSF> 04-Mar-1994 Workspace column headings - the xxx names are used-->
<!--! for backward compatibility with existing .tc_env files-->
<key id="private_object_string" scope="private">オブジエクト</key>
<key id="private_id_label" scope="private">ID</key>

<!-- Part Object -->
<key id="is_designrequired">デザイン痰必覿</key>
<key id="make_or_buy">製造/購買</key>
<key id="part_make">製造</key>
<key id="part_buy">購買</key>
  .
  .
  .
</textsrv>
```

Sample files are provided for each of these configuration files in the **example\bbworkspace\configurations\example** directory.

Migrate a managed supplier site to an unmanaged site

You can change a supplier site that has previously been defined as a managed site in Teamcenter to an unmanaged site without causing any data errors for previously transferred data. This procedure designates the site migrating to an unmanaged site as the supplier site and the OEM site as a managed site.

1. At the supplier site, identify all assemblies that were sent to the OEM site.

2. Ensure that any shared data is not checked out to a remote site.

3. Create new revisions for all shared items:

    a. Turn off the smart baseline feature by changing the value of the **ITEM_smart_baseline** preference to **0**.

    b. At the supplier site, create a baseline of the supplier data that has been sent to the OEM site.

    All new revisions occur in the migrated unmanaged site. Shared objects have multiple revisions, and OEM replica parts are not baselined.

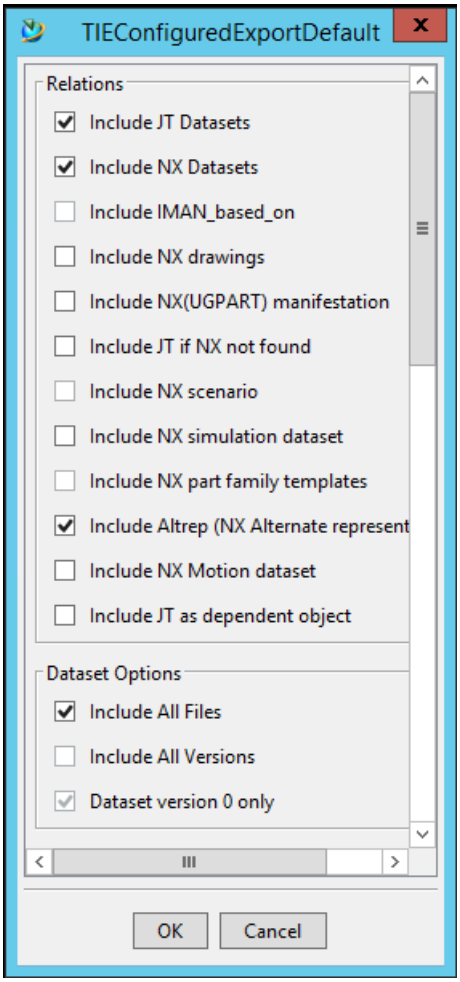
4. If the supplier site is a hub site, in the Organization application, clear the **Is a Hub** check box for the supplier site.

5. At the supplier site, export all supplier-owned objects to a Briefcase file.

    a. Choose **Tools→Export→To Briefcase**.

    b. Select the unmanaged site and click the **Display/Set remote export options** button .

    c. In the **TIEConfiguredExportDefault** dialog box, select the **Replica Bypass** check box.



6. Export all baseline assemblies for migration. This includes:

   - New baseline assemblies that were previously sent.
   - Supplier-owned data.
   - OEM replica data that must be preserved.

7. Install Briefcase Browser at the supplier site being migrated using the same site ID as the managed site for the unmanaged site.

8. In Briefcase Browser, extract all Briefcase files in the migration working directory.

9. Confirm that the Briefcase files extracted meet the OEM requirements.

   Notify the Teamcenter administrator at the OEM site that the Briefcase file output is as expected.

10. In the Organization application at the OEM site, modify the previously managed supplier site by selecting the **Unmanaged** option for the site.

11. At the unmanaged supplier site, use Briefcase Browser to add parts to new assemblies. You can copy the required **.prt** files to a new working directory and create Briefcase files based on them.

12. Transfer the Briefcase files to the OEM and import them into Teamcenter using **Tools→Import→From Briefcase**.

Importing data from earlier versions of Teamcenter

When you are importing data from an earlier version of Teamcenter to the current version and you are not using the site consolidation default import transfer mode (**SiteConsolidationDefaultTMImport**), you may encounter the following error.

```
A relation cannot be saved because it violates a unique index)
```

This error occurs because the TC XML import file is not being properly translated.

Using the **SiteConsolidationDefaultTMImport** transfer mode causes Teamcenter to translate the file for you automatically during the import process. If you use custom transfer modes to import data, you must attach the **Mapping_of_copy_stable_id.xsl** style sheet to the custom transfer mode to translate the import file during the import process. This file is located in the *TC_DATA* directory. Use the **plmxml_tm_edit_xsl** utility to attach the style sheet to the transfer mode.

Using low-level TC XML to improve Briefcase performance when exchanging data between managed Teamcenter sites

Low-level (LL) TC XML allows fast streaming of data for scenarios that move large amounts of data, for example, site consolidation and legacy data migration. Low-level TC XML processes data much faster than other transfer methods. Transfer option sets are configured to use low-level TC XML by default. Older transfer option sets can also be configured to use low-level TC XML by using transfer option sets configured to use low-level TC XML. For information on working with transfer option sets, see What are transfer option sets? in the *PLMXML Export Import Administration Guide*.

Use the following transfer option sets to leverage low-level TC XML with Briefcase:

- **TIEUnconfiguredLLBCZExportDefault** for unconfigured Briefcase export using TC XML.
- **TIEConfiguredLLBCZExportDefault** for configured Briefcase export using TC XML.
- **TIELLBCZImportOptionSetDefault** for Briefcase import using TC XML.

You can further improve the performance of configured Briefcase exports for which you do not need to synchronize the objects being exported. Do so by adding the **opt_no_sync** option to the transfer option set and setting its value to **true**.

When using these transfer option sets, be aware of the following items:

- Anchors are exported and imported.
- **PSOccurrence** threads are exported and imported.
- All **Dataset** revisions are exported and imported.
- Inactive **ItemRevisions** are not exported or imported.

Updating objects previously exported using high-level TC XML

If the target site contains objects that were previously exported using high-level TC XML, some objects may not be updated upon import by default. To import these objects, add the option `forceUpdate=True` to the **TIELLBCZImportOptionSetDefault** transfer option set. Doing so forces the updating of all objects in the TC XML file.

Exporting and importing replica objects

When exporting and importing replica objects (objects for which the owning site is not the local site):

- By default, if an object is a replica, the object will be exported as a stub instead of a full object in the TC XML file. To export the object instead of a stub, set the option `forceExportReplica=True` in the transfer option set.
- When importing objects that already exist at the target site, but the target site is not the owning site, the last save dates (lsd) of the objects are compared. If the lsd of the object in the TC XML is more recent than that of the existing object at the target site, the object is updated. If the lsd values are the same, or if the lsd of the object in the TC XML is older, the object is not updated at the target site.

Using low-level TC XML with custom transfer option sets

To leverage low-level TC XML with custom transfer option sets, add the following option to the option sets:

```
opt_ll_bcz=True
```

Automatically importing items with duplicate ID values

By default, if an item in a Briefcase file has the same ID value as an existing item at the target site, an ID conflict is reported and the item with the duplicate ID is not imported to the target site You can configure Teamcenter to automatically resolve duplicate ID values, streamlining the import of Briefcase files.

Enable detection and handling of duplicate IDs by setting the transfer options **opt_process_conflict_item_id** and **opt_check_conflict_item_id** to **true** in the transfer option sets you are using. When these options are set to **true**, a new **item_id** value is automatically generated for the item with the duplicate ID on import.

If a duplicate ID value is for a Multi Field Key (MFK) item that uses an ID key other than **item_id**, a new **item_id** value is generated for the item and the ID key value is unchanged.

Information about all ID conflicts is included in the import report. ID conflicts are also reported when performing dry run imports and when validating the contents of Briefcase files.

Improving export performance

You can improve performance of certain exports by setting transfer options.

- You can improve the performance of configured Briefcase exports for which you do not need to synchronize the objects being exported. Do so by adding the **opt_no_sync** option to the transfer option set and setting its value to **true**.
- If delta exports are not required, set the **createIxr** option to **False**.
- If your exports do not require site checkout, set the **opt_skip_sco_checks** option to **True**.
- If your exports do not require ACL checks, set the **skipPrivilegeCheck** option to **True**.
- If your exports do not require the infer delete feature, set the **inskipAddExcludedGRMs** option to **True**.

Transfer data between sites that use different schemas

Sites using different schemas may use different objects to represent the same data. For example, you may need to map Site 1's use of Item to Site 2's use of Design. If you are transferring data between sites using different schemas, use the Advanced Multi-Schema Exchanger to create mapping rules to use when transferring data between sites.

Use the following process and examples to as guidance in creating your data mapping.

1. Set up your mapping on Site 2, using the Advanced Multi-Schema Exchanger to map the objects in Site 1's schema to the objects in Site 2's schema.

2. On Site 2, attach the transformation rules to a transfer mode, for example, **Site1ToSite2XferMode**. Attach the transformation rules using a command such as in the following example:

   ```
   plmxml_tm_edit_xsl -transfermode=Site1ToSite2XferMode -action=attach
   -xsl_file=transformerFile
   ```

3. On Site 2, set the **TC_tms_site_interop_transfer_mode** site preference with the values **Site1**, **Site1ToSite2XferMode**.

   When exporting data from Site 1, the objects will be converted based on the rules defined in the mapping rules file attached to **Site1ToSite2XferMode**.

Preventing data loss when sharing data between sites

It is possible to lose data under certain circumstances when mapping data between different systems. An example of this possibility is when the following conditions exist:

- An object is transferred from Teamcenter Enterprise to Teamcenter.
- The object ownership is transferred to Teamcenter.
- The object is subsequently determined to have been transferred in error and the transfer is undone.

This situation could result in the loss of data because some attributes are dropped when an object is transferred from Teamcenter Enterprise to Teamcenter during the mapping processing. This is due to data model incompatibility and because some Teamcenter Enterprise attributes are not needed in Teamcenter (and vice versa). When ownership is transferred back to Teamcenter Enterprise from Teamcenter, data loss could occur. This is because there is no way for the attributes to be repopulated in Teamcenter Enterprise when the attributes were not sent to Teamcenter.

Packaging and exchanging Briefcase files

Understanding the Briefcase transfer process

The process of transferring data for reference using a Briefcase file includes the following steps:

1. **Package the data** as a Briefcase file. Optionally, you can **specify (mark) which objects are to be exported with their ownership transferred** to the target site.
2. **Transfer the Briefcase file** to the destination site through a manual process (such as FTP).
3. Optionally **validate the data in the Briefcase file**, **preview the objects in the Briefcase**, and **compare the objects to objects currently in Teamcenter**. Each of these actions helps you validate the data before importing it at the remote site.
4. **Import the packaged data** at the remote site.

Typically, the owning site (a manufacturer or OEM) initiates the transfer by creating the Briefcase file. The target site (often a supplier) then imports the packaged data. The return trip is similar in that the supplier repackages the data and the originating site imports the Briefcase file.

Mark objects for ownership transfer

Before exporting objects in Briefcase files, you can specify (mark) which objects are to be exported with their ownership transferred to the target site. Marking objects for ownership transfer before the export lets you:

- Precisely identify which objects will have their ownership transferred.
- Verify that the ownership of the objects can be transferred.
- Lock those items against ownership transfer from a different session or by a different user in this same session.

When you mark an object for ownership transfer, the ownership of its entire **island of objects** is transferred upon export. That is, the primary object and the additional objects on which it depends for its correct functional definition and usage within Teamcenter, including item revisions, are owned by the target site upon export. If you mark an item revision for ownership transfer, the full item is marked for ownership transfer.

When ownership of a parent object is transferred, ownership of each of its child **PSOccurrence** objects is also transferred. This allows the target site the ability to edit the parent object without access issues. Ownership is transferred only for direct child objects of the parent object.

Ownership is transferred after the successful export of all revisions of the object, including revisions not explicitly included in the export. You can mark objects of the following types for ownership transfer.

Foundation

- **Item**
- **Item Revision**
- **BOMLine**

4GD

- **Design element**
- **Workset**

To mark objects for ownership transfer, you must be granted the **Transfer out** privilege. For details, see *Access privileges*.

When marking objects for ownership transfer, be aware of the following limitations:

- Objects of the type **Mfg0BvrDynamicIPA** and **Bom0PBELine** cannot be marked for ownership transfer.
- Objects with any of the following statues cannot be marked for ownership transfer:
  - **Remote Object**
  - **Checked Out**

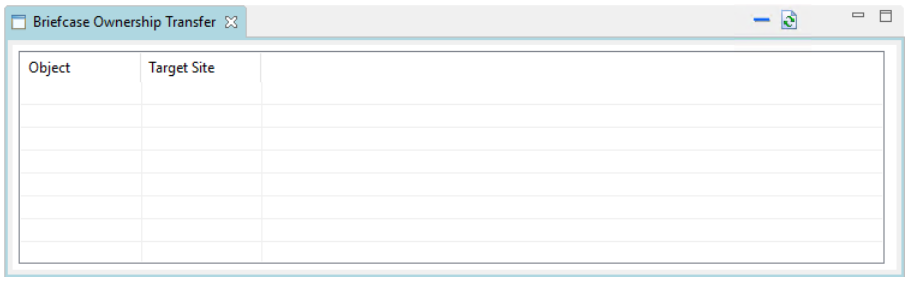
  - **Objects in workflow**

- Avoid marking objects for ownership transfer when performing partial delta exports unless you are sure that the full object will be exported. Objects will not have their ownership transferred if there is no change to an object.

Mark objects for ownership transfer

1. Locate the object to mark for ownership transfer. You can mark only local objects not already marked for ownership transfer.

2. Choose **Window** > **Choose View** > **Other** > **Teamcenter** > **Briefcase Ownership Transfer** and click **Open** to display the **Briefcase Ownership Transfer** view. Drag the object (or objects) to this view.



   Alternatively, select one or more objects, right-click, and choose **Briefcase** > **Mark for Ownership Transfer**.

   The **Ownership transfer to site** dialog box is displayed.

3. On the **Ownership transfer to site** dialog box, select the target site to which you want to transfer object ownership. Only offline sites are available when marking objects for ownership transfer.

   The object is added to the **Briefcase Ownership Transfer** view, which lists all the objects marked for ownership transfer by you, the current user.

4. Once you have marked the objects for ownership transfer, export the objects. Object ownership is transferred when Briefcase export completes successfully. Exported objects are removed from the **Briefcase Ownership Transfer** view.

   Objects that were not part of the export, but were marked for ownership transfer remain listed in the **Briefcase Ownership Transfer** view.

When an object is marked for ownership transfer, users cannot check out the object or mark it for ownership transfer. The object remains locked (including across sessions) until the object is exported or is removed from the **Briefcase Ownership Transfer** view by clicking **Remove selected object from view** .

Package data as a Briefcase file

1. In My Teamcenter, select the part to be added to the Briefcase file.

2. Choose **Tools→Export→To Briefcase** to display the **Export to Briefcase** dialog box.

3. If the desired destination site is not in the **Target Sites** box, click **Select target remote site(s)**  .
   In the **Remote Site Selection** dialog box, add the desired sites from the **Available Sites** to the **Select Sites** list and click **OK**.

4. Select the desired transfer option set from the **Option Set** list.

5. Select the desired check boxes:

Check box	Description
Start immediate transfer	Causes the transfer request to process immediately. By default, transfer requests are scheduled based on system load conditions. Your administrator may disable this option.
Show progress indicator	Displays the progress pane automatically when the process starts.
Send email notification	Sends an email to the address associated with the requesting user when the transfer is complete.
Delta Export	Export only structures modified since the last export.

6. Adjust the selected revision rule if necessary.

7. If you have chosen **a transfer option set configured to use low-level TC XML**, you can analyze the Briefcase TC XML for issues before actually exporting the Briefcase data. Doing so can minimize export and subsequent import errors, reducing the likelihood of needing to re-export data.

   For more information about validating Briefcase TC XML files, see tcxml_export and briefcase_validator.

   Select one of the following additional options:

   Dry Run

   Specifies that a simulated low-level TC XML export be run and the exported TC XML be validated. A report listing warnings and errors in the TC XML is generated and saved to the file specified by **File Name**.

   If **Validate before mapping** is checked in the **Export Options** dialog box, the internal TC XML is validated instead of the exported TC XML.

A Briefcase file is not created in either case.

Validate

Specifies that the exported TC XML be validated. If no warnings or errors are detected in the TC XML, the Briefcase file is created.

If warnings or errors are detected in the TC XML, a report is generated listing them and saved to the file specified by **File Name**. A Briefcase file is not created in this case.

If **Validate before mapping** is checked in the **Export Options** dialog box, the internal TC XML is validated instead of the exported TC XML.

None

Specifies that a Briefcase file is created with no validation of the TC XML.

8. (Optional) Click  to access the advanced options.

> Note:
>
> The **Transfer Ownership** check box is on this secondary dialog box. Therefore, this step is not optional if you want to transfer ownership. This option is not selected by default.

9. In the **Export Options** dialog box, select the desired check boxes:

The options available in this dialog box vary depending on the settings for the selected transfer option set.

Item options

Include All Revisions	Exports all revisions. When transferring site ownership, this is the only option available.
Latest Revision Only	Exports the latest revision regardless of whether it is a working or a released revision.
Latest Any Revisions Only	Exports only the latest revision with the specified release status selected from the list.

Product structure options

Include Entire BOM	Includes all components if the item selected is an assembly. The revision selectors allow you choose the revision to export with the selected item and its component items, if applicable. You can choose only one revision selector.

	The **TC_bom_level_export** preference controls whether this option is available.
Transfer Top-Level Item Only	Transfers the selected assembly item and exports all components with no site ownership transfer.

Dataset options

Include All Versions	Includes all dataset versions with each dataset selected for import or export. When this option is not set, includes only the latest version of each dataset selected for import or export.
Include All Files	Includes all underlying operating system files (such as named references) with each dataset selected for import or export. If you do not set this option, only the dataset metadata is imported or exported.

Save options

Save All Option As Default	Saves the options you select on the dialog box as the default option settings.
Restore Option Set Defaults	Replaces all options settings with the defaults settings.

Session options

Transfer Ownership	Transfers site ownership to the target site. When this option is not selected, your site retains ownership. If you transfer an item revision with a sequence, its sequence manager is also transferred. The **TC_ownership_export** preference controls the default value of this option. It is recommended that you leave the default setting for this option cleared.
Include Modified Objects Only	Exports unmodified objects as stubs. Only modified objects are transferred as full objects. Note: If a change is made to any dependent revision in an assembly, the last modified date of the assembly is updated. Therefore, the assembly and all its dependent revisions are exported as full objects.

Validate before mapping	Validates the internal TC XML instead of the exported TC XML when **Dry Run** or **Validate** are chosen. This option is only available when **Option Set** specifies a transfer option set configured to use low-level TC XML.
Continue On Error	Allows the remote import or export objects operation to continue if errors are encountered while importing or exporting optional objects. All objects are considered optional except the following: **Requirements** **Specifications** **Item Master** **Item Revision Master** This option is disabled if the **Transfer Ownership** option is set.

10. Click **OK** and confirm your settings. Teamcenter sends a Briefcase package file to your (the exporting user's) mailbox.

11. Select the Briefcase package file and transfer the package to the file system.

Transfer a Briefcase file to another site

Teamcenter recognizes both *.bcz* and *.ugs* extensions as Briefcase files.

1. In My Teamcenter, select the Briefcase file from the exporter's mailbox.

2. Choose **View→Named References**.

3. Select the Briefcase file and click **Download**.

4. Click **OK** to export the file to the specified directory. Click **Close** to exit the dialog box.

Use email, FTP, or some other manual method to transfer the Briefcase file to the importing site.

Validate Briefcase files and low-level TC XML data

You can optionally validate the integrity of an existing Briefcase file and the low-level TC XML data in an existing Briefcase file or other low-level TC XML file before importing and committing the data. Doing so reduces the likelihood of needing to reimport data.

Validate the integrity of a Briefcase file using the briefcase_validator command line utility. Validate the integrity of the low-level TC XML data within a Briefcase file or other TC XML file using tcxml_import.

Validate the Briefcase file

1. Open a Teamcenter command prompt.

2. Validate the file using a command of the following form:

   ```
   briefcase_validator -inputfile=file_name
   ```

   Where:

   file_name

   The path and file name of the Briefcase file.

   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.

   By default, **briefcase_validator** lists its results in the command window. You can optionally send the results to an HTML file by using the **-report** option to specify a file with a *.html* extension. See briefcase_validator for details of the command's options.

Example:

Evaluate the exported TC XML in an existing Briefcase file.

```
briefcase_validator -u=admin01 -p=pass01 -g=dba -inputfile=
c:\temp\sample_briefcase.bcz
```

Evaluate a stand-alone low-level TC XML file and send the results to an HTML file for viewing.

```
briefcase_validator -u=admin01 -p=pass01 -g=dba
-inputfile=c:\temp\sample_tcxml.xml -report=c:\temp\report.html
```

Validate the contents of the Briefcase file

Use a combination of the following tools to validate the contents of a Briefcase file:

Tool	Description
Validate the Briefcase file contents using the Teamcenter import options	On the **Import Briefcase** dialog box, select the **Dry Run** and **Validate** options. The **Dry Run** option simulates a low-level TC XML import and identifies any issues in the TC XML.

Tool	Description
	The **Validate** option examines configuration aspects of the Briefcase file along with the integrity of the objects in the Briefcase file.
Validate the Briefcase file contents using the command line import options	When using the **tcxml_import** command, use the **-dryrun** and **-validate** arguments. **-dryrun** simulates a low-level TC XML import and identifies any issues in the TC XML. **-validate** examines configuration aspects of the Briefcase file along with the integrity of the objects in the Briefcase file.
Preview the objects in the Briefcase file	View objects, JT files, CAD files, and related documentation in a Briefcase file before importing them using the **Briefcase Preview** pane.
Compare the objects in the Briefcase file to existing Teamcenter data	Compare objects in a Briefcase file to the objects currently in Teamcenter or to the objects in another Briefcase file to review differences before importing the data.

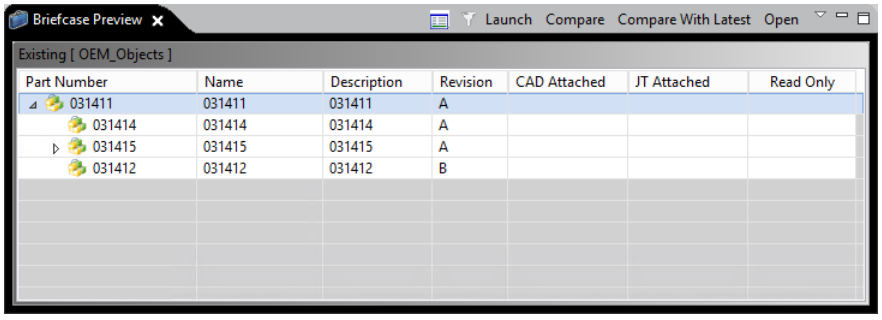
Preview objects in Briefcase files

View the contents of a Briefcase file using the **Briefcase Preview** pane. You can view Briefcase files containing JT files and related documentation and CAD files and related documentation depending on the configuration you have selected for the **Briefcase Preview** pane.

Ensure that Briefcase Viewer has been properly configured and the related configuration files have been copied to the site configuration files directory.

Open and preview a Briefcase package

In My Teamcenter, right-click a Briefcase (**bcz**) file and choose **Briefcase→Open**. The Briefcase file opens in the **Briefcase Preview** pane. The contents of this package are listed as the **Existing** objects.



Expand and collapse the Briefcase hierarchy to review the objects in the package.

Compare objects in Briefcase files

You can compare objects in Briefcase files to review differences before deciding to import changes. You can compare objects in a Briefcase file to the objects currently in Teamcenter or to the objects in another Briefcase file.

Ensure that Briefcase Viewer has been properly configured and the related configuration files have been copied to the site configuration files directory.

Compare Briefcase packages

1. **Open a Briefcase file in the Briefcase Preview pane**. The contents are listed as **Existing** objects.

2. Choose a Briefcase file to compare with the opened Briefcase file:

   - To compare objects in the opened Briefcase file with the objects currently in Teamcenter, click **Compare with Latest**.

     Alternately, in My Teamcenter, right-click the open Briefcase file and choose **Briefcase→Compare with Latest**.

   - To compare objects in the opened Briefcase file with objects in a second Briefcase file, click **Compare**. Navigate to and open the second Briefcase file.

     Alternately, in My Teamcenter, select both files, right-click, and choose **Briefcase→Compare**.

   These objects are listed as **Proposed** objects.

3. Click **Comparison** to view the package differences, identified with color coding. By default:

   - Red identifies objects not in the newer Briefcase file.
   - Green identifies objects not included in the older Briefcase file or stored in Teamcenter.
   - Blue identifies objects that are changed.

   If no differences exist, the top level of the Briefcase package displays no child items.

4. Optionally filter the displayed comparison results when comparing long lists of objects. Click in the **Briefcase Preview** pane and select one of the following filter options:

   No Filter

   All objects in the Briefcase structures are displayed. This is the default and initial setting for every comparison.

   Show Changed Objects

Only those objects that are changed (and any parents) are displayed.

Show Added Objects

Only objects that are added (and any parents) are displayed.

Show Deleted Objects

Only objects that have been deleted (and any parents) are displayed.

Show Changed, Added and Deleted Objects

Only objects that are changed, added, and deleted (and any parents) are displayed.

Click **Cancel** to end the comparison.

Import objects in Briefcase files

1. In My Teamcenter, choose **Tools→Import→From Briefcase** to display the **Import Briefcase** dialog box.

2. Set **Briefcase File** to the Briefcase package file to be imported.

3. Set **Option Set** to the transfer option set containing the configuration options to use during import.

4. Optionally check any desired import options:

Option	Description
Start immediate transfer	Causes the transfer request to process immediately. By default, transfer requests are scheduled based on system load conditions. Your administrator may disable this option.
Show progress indicator	Displays the progress monitor dialog box automatically when the process starts.
Send E-mail notification	Sends an email when the transfer is complete to the address associated with the requesting user.

5. You can **preview** and **compare** objects in Briefcase files to review differences before deciding to import changes. You can compare objects in a Briefcase file to the objects currently in Teamcenter or to the objects in another Briefcase file.

   If you have chosen **a transfer option set configured to use low-level TC XML**, you can analyze the Briefcase TC XML for issues before actually importing the Briefcase data. Doing so can minimize import errors, reducing the likelihood of needing to re-import data.

   For more information about validating Briefcase TC XML files, see tcxml_import and briefcase_validator.

6. Select one of the following additional options:

Dry Run

Specifies that a simulated low-level TC XML import be run and the imported TC XML be validated. A report listing any warnings and errors in the TC XML is generated and displayed. The report is also saved in the same directory as the Briefcase file, named the same as the Briefcase file, but with a *.html* extension.

Optionally click  and check **Validate before mapping** to validate the exported TC XML before it is transformed.

In neither case is the Briefcase file imported.

Validate

Specifies that the imported TC XML be validated. Two levels of validation occur:

- In the first validation level, configuration aspects of the Briefcase file such as the user, transformation rules, data types, and the target site are validated.
- In the second pass, the objects in the Briefcase file are validated.

If errors are found in the first pass, the Briefcase file is not imported. If errors are found only in the second pass, the Briefcase file is imported. Whenever warnings or errors are found, a report is generated and displayed. The report is also saved in the same directory as the Briefcase file, named the same as the Briefcase file, but with a *.html* extension.

Optionally click  and check **Validate before mapping** to validate the original TC XML instead of the transformed TC XML.

None

Specifies the Briefcase file is to be imported with no validation of the Briefcase file.

7. Click **OK** and confirm your import settings. The objects in the Briefcase file is imported.

Recover object ownership transferred in Briefcase files

You can regain the ownership of Briefcase objects from other sites when that ownership has inadvertently been transferred or is no longer valid.

Recover transferred ownership by running the **briefcase_ownership_recovery** utility. Ownership can be recovered only in cases where ownership was previously transferred using a release of Teamcenter that includes the **briefcase_ownership_recovery** utility.

The **briefcase_ownership_recovery** utility reverts the ownership of the objects associated with a Briefcase to the original (exporting) site. The objects are reverted to replicas on the remote (importing) site.

> Caution:
>
> To ensure data integrity, run this utility at both the exporting site and the importing site.

You can review the objects for which ownership can be changed with the utility's **-report** option before actually implementing the ownership changes.

Export Briefcase files from the command line

For test and scripted batch operations, you may find it more efficient to export Briefcase files from the command line. The **tcxml_export** utility offers an extensive set of options to control your export operation. You must have Teamcenter administration privileges to run **tcxml_export**.

Command line export is supported only when exporting to managed sites.

Use the following examples as guidance for common export scenarios. See tcxml_export for details on all of the options available when exporting data using the command line.

Run the following commands from a Teamcenter command prompt.

Export an item to a Briefcase file

To export a specific item to a Briefcase file, use a command with the following form:

```
tcxml_export -briefcase -file=file_name -optionset=optionset_name
-targetsites=site_ID -item=id
```

Where:

file_name

The path and file name of the created Briefcase file.

optionset_name

The transfer option set name used to export the objects.

site_ID

The ID of the target site(s). Target sites must be managed sites.

id

The **item_id** value of the item to be exported. Specify multiple items by separating values with colons.

Transfer object ownership to another site

You can use **tcxml_export** to export objects with their ownership transferred to the target site with using the **objsForOwnXfer** session option. Ownership transfer has the same limitations described in

Mark objects for ownership transfer and works in a similar manner. One difference when exporting from the command line is you do not need to mark the objects with ownership being transferred before running **tcxml_export**.

Use a command with the following form:

```
tcxml_export -briefcase -file=file_name -optionset=optionset_name -item=id
-targetsites=-site_ID -session_options=objsForOwnXfer:-site_ID:item_uid
```

Where:

site_ID

The ID of the target site. When transferring ownership, only one target site ID is allowed.

item_uid

The unique identifier (UID) of the item for which ownership is transferred.

Validate the TC XML to be exported

You can validate your exported TC XML and create a briefcase file only if no warnings or errors are encountered. If errors or warnings are encountered, a report file is created. Use a command with the following form:

```
tcxml_export -briefcase -file=file_name -optionset=optionset_name -item=id
-targetsites=-site_ID -validate
```

Where:

file_name

The path and file name of the created Briefcase file. If errors or warnings are encountered a report file is created with this file name and a *.html* extension.

Consider using the **-dryrun** option in place of the **-validate** option to validate the TC XML without actually exporting the data.

Import Briefcase files from the command line

For test and scripted batch operations, you may find it more efficient to import Briefcase files from the command line. The **tcxml_import** utility offers an extensive set of options to control your import operation. You must have Teamcenter administration privileges to run **tcxml_import**.

Use the following examples as guidance for common import scenarios. See tcxml_import for details on all of the options available when importing data using the command line.

Run the following commands from a Teamcenter command prompt.

Import the contents of a Briefcase file

Use a command with the following form:

```
tcxml_import -u=Tc-admin-user -p=password -g=group
-briefcase -file=file_name -optionset=optionset_name -briefcase
```

Where:

Tc-admin-user, *password*, *group*
: Credentials of a Teamcenter administrator.

file_name
: The path and file name of the Briefcase file.

optionset_name
: The name of the transfer option set containing the options to use during import.

Analyze Briefcase files before importing them

You can quickly view and validate the low-level TC XML data in an existing Briefcase file using the **briefcase_validator** utility. Doing so before actually importing and committing the data minimizes import errors and reduces the likelihood of needing to re-import data. See briefcase_validator for details on all of the options available when using validating data for import using the command line.

Use a command with the following form:

```
briefcase_validator -u=Tc-admin-user -p=password -g=group -inputfile=file_name
```

Where:

file_name
: The path and file name of the Briefcase file to be validated.

  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.

Consider using the **-dryrun** or **-validate** options as alternate methods of validating the TC XML.

Using remote site checkout and checkin to allow other sites to update objects

Check out objects to a remote site

You can check out an object to a site (such as a supplier site) so that it can be modified by the site. This process creates a Briefcase package file that you can transfer to the site where it can be imported. It

also creates a reservation object and associates it with the checked-out object. You cannot check out an object to a site if it is already checked out either locally or to another site. If you log on to a hub site, you can also check out a replica object to site.

Site checkout functionality is intended for collaboration between OEMs and suppliers for the following purposes only:

- Adding or modifying datasets at a supplier site.
- Adding or modifying components (**ps occurrences**) at a supplier site.
- Modifying metadata (attribute values) of workspace objects at a supplier site. Modification of non-workspace objects and revising item and item revisions is not supported.

To check out an object to a remote site:

1. In My Teamcenter, select the object you want to check out.

   You can perform site checkout on the following objects and their subclasses:

   - **Item**
   - **ItemRevision**
   - **PSBOMView**
   - **PSBOMViewRevision**
   - **Form**

   Site checkout of **Schedule**, **ScheduleRevision**, **ScheduleTask**, **ScheduleTaskRevision** objects, and their related objects is not supported.

2. Choose **Tools→Site Check-In/Out→Check-out To site**.

3. In the **Check-out To Site** dialog box, choose the remote site from the list of target sites. Optionally, type a change ID and comments in the respective fields.

4. Click **Explore Selected Component(s)** , select the related objects to check out, and click **OK**.

5. Click **Yes**.

Check in objects from a remote site

After you **import a briefcase package** that has **objects checked out to a site**, you can check in the objects to your site. You can also check in a replica object.

1. In My Teamcenter, select the objects that you want to check in to your site.

2. Choose **Tools→Site Check-In/Out→Check-in From site**.

3. Click **Yes**.

When an object is checked in to a site:

- If it was checked out to another site, the reservation object is restored to its precheckout state.
- If the object was not checked out to another site, the reservation object is removed.
- The export record is updated.

Ownership of new objects added at supplier site is determined based on the following criteria:

- New objects created at a supplier site that are part of an existing island from the OEM are owned by the OEM. For example, if a dataset or version is added to an **Item** or **ItemRevision** object at a supplier site that is owned by the OEM, the new dataset or version is also owned by the OEM.
- New objects created at a supplier site that are part of a new island at the supplier site are owned by the supplier. For example, if an **Item** object (component) is added at a supplier site to an assembly owned by the OEM, the new object is owned by the supplier.

Islands are logical groups of objects where the primary object is an item. For example, the following objects are part of an island.

- **Item** (primary object of an island)
- **ItemRevision**
- **Dataset**
- **Form**
- **Relation**
- **PSOccurrence**

Cancel the remote-site checkout of objects

If you checked out an object to another site, you can cancel the checkout by choosing **Tools→Site Check-In/Out→Cancel Check-Out to site**. You can also cancel checkout of a replica object. Only an administrator can cancel a site checkout when another user performed the checkout.

When a site checkout is canceled:

- If it was checked out to another site, the reservation object is restored to its precheckout state.
- If the object was not checked out to another site, the reservation object is removed.
- The export record is removed.
- Datasets are not restored to their precheckout state. The only action is the site checkout lock is removed.

Exchanging Briefcase packages with unmanaged sites

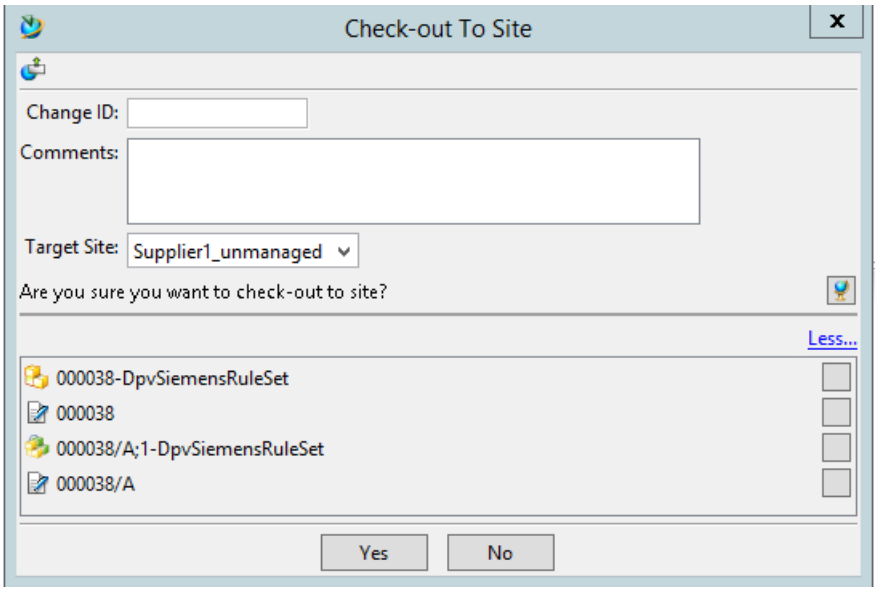
Export objects to unmanaged sites using Briefcase

1. Log on to Teamcenter and open the My Teamcenter or Structure Manager application. Locate and expand the top level of the assembly structure that you want to export.

2. For each object that the unmanaged site must update (but not own locally), select the object and choose **Tools→Site Check-In/Out Check-out To Site**.

   > Tip:
   >
   > Because the types of non-CAD datasets you may want to export are various, for performance reasons, unmanaged site exports do not have support for any specific non-CAD datasets by default. To include non-CAD datasets in the export, add a clause to the transfer options set you are supporting.
   >
   > For example, to add support for Microsoft Word (**MSWORD**) datasets to your export, include the following clause in the closure rules for the transfer option set:
   >
   > ```
   > ItemRevision:MSWORD:Iman_reference.*
   > ```

   Teamcenter displays the **Check-out To Site** dialog box.

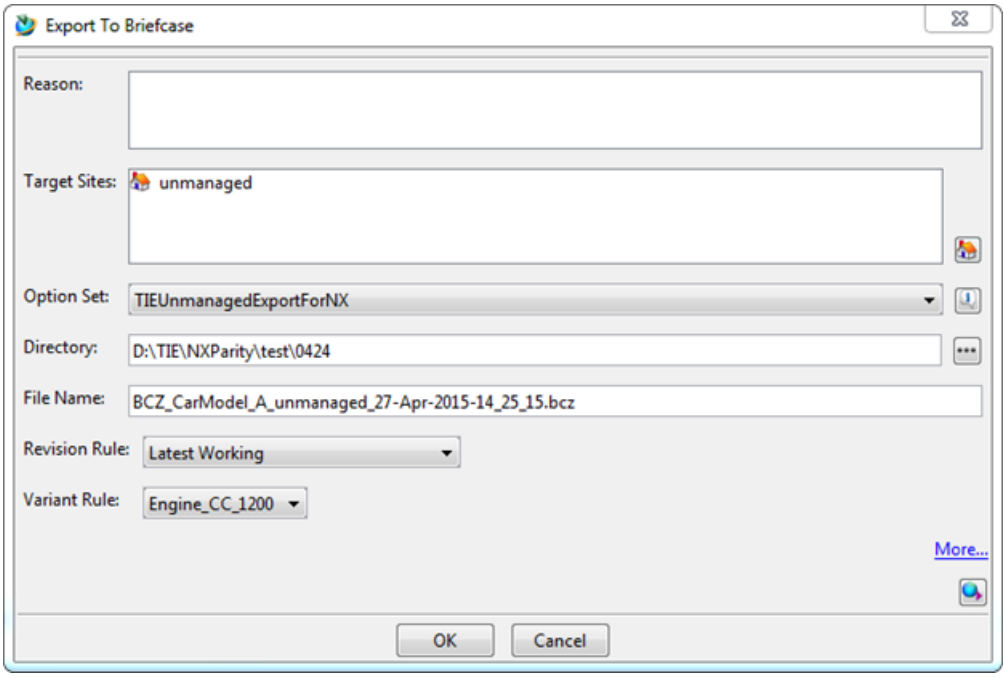


3. Select the unmanaged site from the **Target Site** list.

4. (Optional) Type an identifier in the **Change ID** box and a comment in the **Comments** box.

5. Select the item revision associated with the top level of the assembly structure that you want to export and choose **Tools→Export→To Briefcase**.

   Teamcenter displays the **Export to Briefcase** dialog box.



6. (Optional) Type a reason statement in the **Reason** box.

7. Click the site button and select the target site.

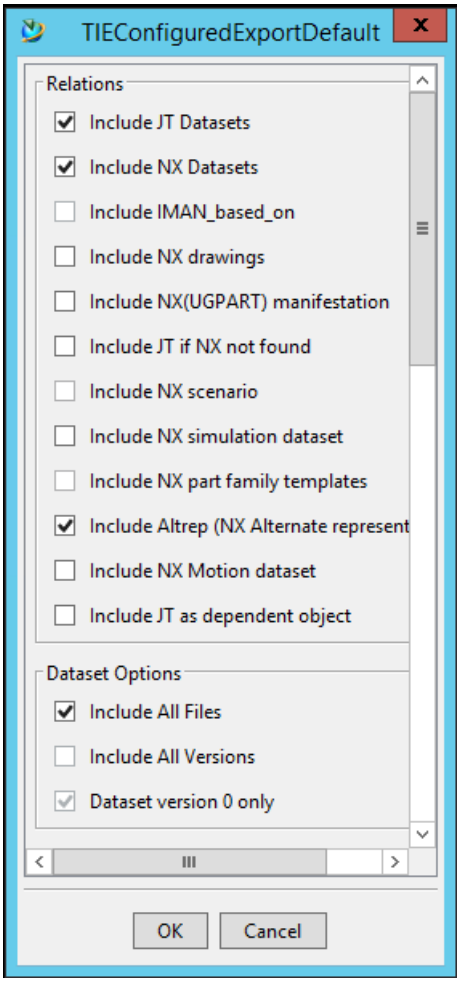
8. Select the option set you want to use for this transfer from the **Option Set** list. Select the standard option (**TIEConfiguredExportDefault**) for configured exports or one of the other available option sets. For transfers containing NX data, select **TIEUnmanageExportForNX**.

9. Enter the desired directory in the **Export Directory** box. This is the location where you can access the generated Briefcase file for transfer to the unmanaged site.

10. Select the desired revision from the **Revision Rule** list.

11. Click **Display/Set export options** .

    Teamcenter displays the *option-set-name* dialog box.



Unmanaged sites do not have the ability to manage dataset versions. The latest version, and only the latest version, is required. Therefore, the **Dataset version 0 only** option is selected by default and cannot be changed.

If you select the **Include modified objects only** session option, unchanged objects are exported as stubs, unless a new or changed dependent object appears under the selected root object. The export process recognizes a change to the **lsd** attribute of the item, causing a full export of all revisions.

> Note:
>
> If you are transferring an assembly with at least one part marked for ownership transfer to the importing site (not the entire assembly), you must transfer the parts that have ownership transferred in a separate Briefcase file. Otherwise, you must create closure rules to prevent the transfer of ownership of other parts in the assembly to the importing site.

12. Click **OK** and click **Yes** to begin the export.

    Teamcenter displays the **Export Completed** dialog box when the export completes.

Import validation results

Use the validation rule editor to create a rule set used to create a validation results (VRX) file.

1. From your Teamcenter command shell, run the **ruleseteditor.bat** file:

   ```
   %UGII_BASE_DIR%\DESIGN_TOOLS\checkmate\tools\ruleset_editor
   ```

2. In the Validation Rule Editor, create a new rule set with the **checker** attribute value set to **mqc_update_all_features** and the **Dataset Types** attribute value set to **UGMASTER**.

3. Save the file.

4. In Teamcenter, create a validation rule set dataset and import the rule file you created into the new validation rule set.

5. Open options and locate the **BRIEFCASE_import_validation_rule_item** preference. Set its value to the item ID or revision ID of the item you want to validate.

6. Import the Briefcase file and verify that the validation forms for the imported item revision are present.

Import a Briefcase file from an unmanaged site

1. Log on to Teamcenter and open the My Teamcenter or Structure Manager application and choose **Tools→Import→From Briefcase**. The **Import Briefcase** dialog box is displayed.

2. Enter the path and file name of the Briefcase file and set **Option Set** to **TIEImportOptionSetDefault**.

3. (Optional) Set any of the following options:

Start immediate transfer	Starts the import to process immediately. (Your administrator can disable this option.)
Show progress indicator	Opens your default web browser and displays the import steps as they occur.
Send E-mail notification	Sends an email to the email address associated with the current user when the import is complete.

4. (Optional) Click **Display/Set export options** and set any desired options.

5. Click **OK** and confirm your import settings to start the import.

Exporting bills of materials (BOMs)

Exporting a configured bill of materials (BOM)

You can export a configured BOM from My Teamcenter. You can also export a configured BOM from a command line using the **tcxml_export** utility. You must be a user with administrator privileges to use the utility.

The following transfer options are required:

Transfer option	Description	Default value
opt_exp_cfgbom	Starts the configured BOM export and controls the traversal flow to prevent the export of persistent objects related to lines that are configured out.	**False**
opt_rt_only	Controls whether the exported data contains only runtime objects or both runtime and related persistent objects.	**False**
opt_exp_cnf	Exports the variant configuration data when the structure is configured by only a revision rule (no variant rule is applied). A structure configured this way contains more lines than are found in a realizable product (150 percent BOM).	**False**
opt_exp_persistentobj_only	Blocks serialization of runtime objects in the TC XML file. Configured export and import using high level TC XML is supported only for versions 8.3.3 and 9.1 and later. Serialization of runtime objects must be turned off for prior versions of Teamcenter.	**False**

The following session options are required:

Session option	Description
revRule	Specifies the name of the revision rule to use. If a configured BOM export is started without this session option defined, the revision rule is read from the **TC_config_rule_name** preference.
varRule	Specifies the name of the saved variant rule or stored option set. Supports classic, modular, and hybrid variants. The values that Teamcenter displays for the variant rules are saved in the database. Be aware that if the revision on which the variant options are populated is not part of the BOM being exported, the following closure rules must be added for the revision to be exported:

Session option	Description
	CLASS.Variant:CLASS.Item:ATTRIBUTE.parent_item:PROCESS+TRAVERSE CLASS.Item:CLASS.PSBOMView:ATTRIBUTE.bom_view_tags:PROCESS CLASS.ItemRevision:CLASS.PSBOMViewRevision:ATTRIBUTE.structure_-revisions:PROCESS

The export of a configured BOM using incremental change (IC) is covered as part of the revision rule configuration. Additionally, you can control the visibility of components in the constructed BOM with the following options:

Session option	Description	Default value
processUnconfiguredByOccEff	Controls export of the **BOMLine** object configured out by occurrence effectivity.	**False**
processSuppressedOcc	Controls export of suppressed **BOMLines** objects.	**False**
processUnconfiguredVariants	Controls export of unconfigured variants.	**False**
processUnconfiguredChanges	Controls export of **BOMLine** objects configured out by incremental change.	**False**

Exporting incremental changes that are applied in context of a sublevel is not supported. Changes must be done by selecting the topline context explicitly.

Synchronizing nonworkspace objects

When synchronizing nonworkspace objects using high-level configured TC XML, Teamcenter exports full objects even when you select the modified objects only option. You can use low-level TC XML configured export as an alternative.

If you select the modified objects only option during an unconfigured export, Teamcenter determines the objects requiring synchronization based on the correct parents IXR, resulting in export of only the modified objects.

However, when there are no physical structure changes, both high-level and low-level TC XML export functions cannot identify the synchronization candidates, causing full objects to be exported. An example of this situation is when only effectivity or revision rule changes have been made.

Exporting a BOM without variant rules

A structure that is configured by only a revision rule (no variant rule is applied) contains more lines than are found in a realizable product resulting in a 150 percent BOM. The exported TC XML file must contain enough information for the target site to solve the variant conditions. To accomplish this, the TC XML file contains the configuration data stored in the following variant data model classes:

- **VariantExpression**
- **VariantExpressionBlock**
- **ConfigurationCNF**
- **ConfigurationExprLiteral**
- **ConfigurationFamily**

Legacy and classic variants are supported; modular and hybrid variants are not supported. You must select the **opt_exp_cnf** transfer option (set to **True**) for this type of export.

Working with BOM changes (delta export)

BOMLine objects can be updated by many workflows and actions such as:

- **BOMLine** updates in Structure Manager
- Computer aided design (CAD) application updates to occurrences and end items
- Revise, baseline, release, check out and check in operations
- Multi-Site import and export operations such as replica updates

Because **BOMLine** objects are runtime objects, updates must be sent on a regular basis to provide the proper representation of a BOM in the data caches. To accomplish these updates efficiently, Teamcenter provides the ability to:

- Assign a stable identity to runtime objects that have been exported. The identity is stored in the accountability table.
- Express a **BOMLine** object as a function of certain persistent Teamcenter objects (a recipe) that are stored in the recipe table.
- Identify changed (dirty) **BOMLine** objects by combining the date in the recipe, accountability, and scratch tables. Edits to Teamcenter data are stored in the scratch table.
- Reconfigure changed **BOMLine** objects by using their occurrence thread chain. These changes are stored in the scratch table.

Unless you specify a root object as an input for the **tcxml_export** utility, a delta export contains all changes to all exported configured structures. No specific root object is used. To specify a root object, you must use the **-inputfile** or **-inputduidfile** arguments with the **-sync** argument.

The process for a delta export of a specific configuration is described in *Synchronize changes for a specific BOM configuration*.

All applications that use TC XML and perform data synchronization have their application IDs registered in the Teamcenter database's subscription table (**SUBSCRIPTION_TABLE**). If an application ID is inactive longer than the time specified by the **install** utility's **TC_TIMESTAMP_THRESHOLD** parameter, it is deleted from the subscription table. When this happens, the related data is no longer synchronized. The default value of **TC_TIMESTAMP_THRESHOLD** is 96 hours

See Maintain data synchronization through site consolidation for more information.

Synchronize changes for a specific BOM configuration

This process requires the **-low_level** argument of the **tcxml_export** utility. Therefore, you must set the **SITCONS_AUTH_KEY** environment variable in your command environment to a valid license value. Open a support case on Support Center to get this license value.

Delta export for a specific BOM configuration requires a root object used to reconfigure and export the modified component lines for the BOM configuration.

1. Export an assembly with specific configuration rules:

   - **Revision rule**

     Latest Working Only

   - **Variant rule**

     VRule1

     BOMWindow

   - Export options

     Export suppressed bomlines

     ```
     tcxml_export -u=tcadm -p=admpw> -g=dba -low_level
     -inputfile=d:\temp\inp.txt-svrule=VRule1 -processSuppressedOcc
     -optionset=TIEConfiguredExportDefault_LL -file=d:\temp\out.xml
     ```

2. In Teamcenter, make changes to the product structure such as adding, deleting, or modifying child components.

3. Delta export the configured assembly by specifying the top item ID and the same configuration rules as the initial export.

```
tcxml_export -u=tcadm -p=admpw> -g=dba -low_level -sync
-inputfile=d:\temp\inp.txt -svrule=VRule1 -processSuppressedOcc
-optionset=TIEConfiguredExportDefault_LL -file=d:\temp\out_sync.xml
```

Repeat a previous export of a specific BOM configuration

You can repeat the most recent export of a particular export configuration. Doing so helps address situations such as:

- Importing a delta export failed and you need to regenerate the same delta export package.
- You need to correct an issue with the data and then recreate the previous delta export, including the updated data and unchanged data from the previous delta export.
- Other situations where repeating the previous delta export is more efficient than manually updating exported data.

You can re-export configured exports of high level and low level TC XML to managed or unmanaged sites. Full and partial BOM exports are supported.

1. In the Teamcenter rich client, choose the same root object as used in the previous export you wish to recreate.
2. Choose **Tools** > **Export** > **To Briefcase**. The **Export to Briefcase** dialog box is displayed.
3. Choose the same target site, revision rule, and variant rule as used in the previous export.

   You do not need to mark objects for ownership transfer. The same ownership transfer specifications will be used as in the previous export. Objects with their ownership transferred in the previous export will also be included in this export.
4. Check **Force Re-Export** and click **OK**. The most recent export meeting this configuration is repeated.

   If the previous export meeting this criteria was a full export, the full export is repeated, including changes since the previous full export. If there is no previous export, a full export is created.

Export a bill of materials to a Briefcase file

Use the following steps to export a configured bill of materials (BOM) to an offline site as a Briefcase file without having middleware (such as Teamcenter Integration Framework) installed.

1. To export a BOM from My Teamcenter, select the BOM and choose **Tools→Export→To Briefcase**.
2. In the **Export To Briefcase** dialog box, click **Select target remote site(s)** ( ) and select the site desired remote sites.

3. Select **TIEConfiguredExportDefault** from the **Options Set** list.

4. Click **Display/Set export options** ( ) to display the **TIEConfiguredExportDefault** dialog box.

   Select the desired export options.

5. Select the desired revision rule form the **Revision Rule** list.

6. Select the desired variant rule form the **Variant Rule** list.

7. Select or type the desired directory in the **Export Directory** box and type the desired file name in the **Export File Name** box.

8. (Optional) Type a description of the export in the **Reason** box.

Sharing 4GD data using PLM XML

Sharing 4th Generation Design data

4th Generation Design (4GD) is the process of developing a collaborative design that contains all the necessary design information used to manufacture the subject product, for example a car, ship, or aircraft. A collaborative design is the collector for product configurations and includes all product design data as design elements and design features. You develop your collaborative design using the 4G Designer application. 4G Designer supports massive product designs with millions of business objects along with enhanced collaborative business processes.

Contact your Siemens Digital Industries Software representative for more information on working with 4GD data.

You can exchange 4GD data by exporting and importing Briefcase files. Briefcase files are designed to handle the massive data required for 4GD data exchange.

Forward and backward compatibility between Teamcenter versions requires the Teamcenter schemas to be compatible at both the exporting and importing sites. Therefore, you cannot export 4GD data from earlier Teamcenter versions that do not support the 4GD data model.

> Note:
>
> The Briefcase Browser application does not support briefcase files containing 4GD data.

You can export and import configured 4GD data and its supporting infrastructure in an unmanaged site briefcase file. A configured export to and unmanaged site can include JT datasets but cannot include NX related datasets.

4GD business object support

All 4GD business objects are supported in Briefcase replica transfers. The following 4GD objects can be selected as root objects:

- Collaborative design (**Cpd0CollaborativeDesign** class)
- Design control element (**Cpd0DesignControlElement** class)
- Design feature (**Cpd0DesignFeature** class)
- Workset (**Cpd0Workset** class)
- Promissory, reuse, and shape design elements (**Cpd0DesignElement** class)
- Partition (**Ptn0Partition** class)
- Subset definition (**Mdl0SubsetDefinition** class)
- Partition template model (**Ptn0PartitionTemplateModel** class)

Exported objects are listed as the primary object in the export log. The imported objects are listed as new objects in the import log.

When importing the same object after the initial import, the imported object is listed as updated in the import log.

For roots objects designated as not exportable, the object is listed as **STUB_INSUFFICIENT_PRIVILEGE** in the export log and a stub object is created at the importing site.

The following TC XML options apply to 4GD object replication.

TC XML option name	Description	Export behavior
opt_de_rlz_item	By default, BOM structure is exported separately from 4GD structure due to the potential size of the data. If you want to export BOM structure and 4GD structure together in single operation, set this option to **true**. BOM structure includes the full island of data of an associated item/BVR with a design element.	**DesignElement** source objects are exported as full objects. For **ShapeDE** objects, **ShapeDesignRevision** is the source object. The **ShapeDesignRevision** object is always exported with the **ShapeDE** object even if the **opt_de_rlz_item** option is set to **FALSE**. For **ReuseDE** and **SubordinateDE** objects, **ItemRevision** is the source object.
opt_workset_rlz_de	By default, the **Workset**/**Subset** structure is exported separately from **DesignElement** objects. If you want to	Realization map objects (**Rlz0ModelRealizationMap**) are exported as full objects. The

TC XML option name	Description	Export behavior
	export the **Workset**/**Subset** structure and **DesignElement** objects together in a single operation, set this option to **true**.	source object can be a **DesignFeature**, **DesignElement**, or **DesignControlElement** object.
opt_exp_cfgbom	Sets the export mode for either configured or unconfigured 4GD data, Set to true to export configured 4GD data.	Configured and unconfigured behavior are explained following this table.

Partition data models

You can add a design element under a replicated partition. The primary partition does not have the knowledge of all design elements under the replicated partition. This behavior is acceptable in the following circumstances:

- When a well-defined best practice approach is exercised so your users can assume the primary partition is always up to date within a given time interval.
- When a central authority, implemented in the future, can be consulted to determine what data is up to date.

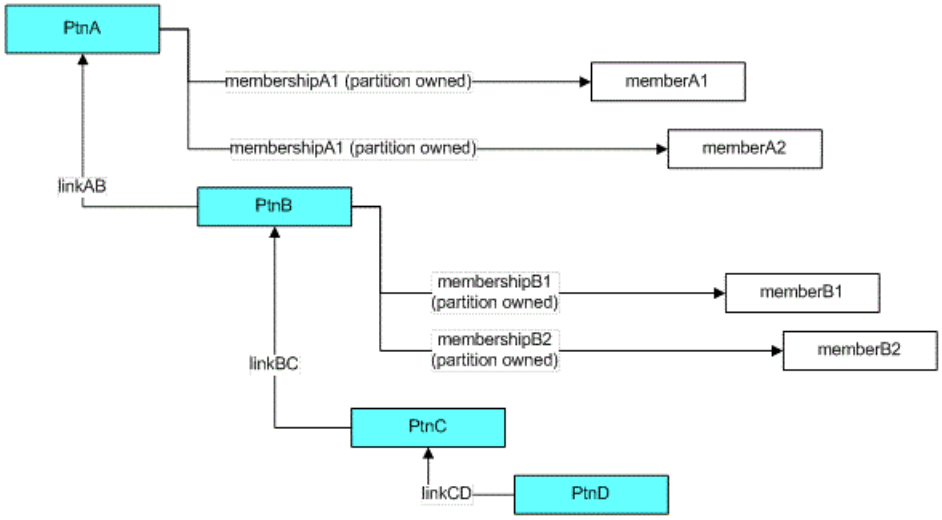
For a parent-child link owned by child partitions, users must make sure the partition hierarchy is constructed at one site and then replicated to all other sites.

When replicating partition data model memberships:

- Partition-owned memberships are exported along with the partition that owns them.
- Member-owned memberships are exported along with the member that owns them.

When replicating partition data model members:

- Member-owned memberships are exported as full objects.



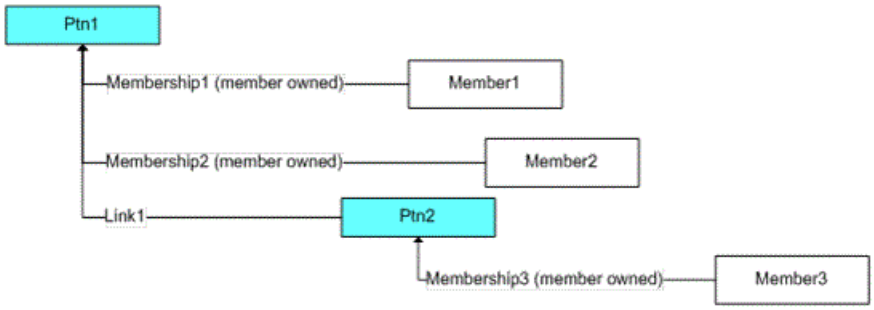
- If **PtnB** is the root object:

  LinkAB, **membershipB1**, **memebershipB2**, and **PtnB** objects are exported as full objects.

  The **memberB1** and **memberB2** objects are exported as full objects.

- If **PtnC** is the root object, the **PtnC** and **linkBC** objects are exported as full objects.

- Member-owned memberships are exported along with the member that owns them.



- If **Ptn1** is the root object, only the **Ptn1** object is exported.
- If **Ptn2** is the root object, the **Link1** and **Ptn2** objects are exported as full objects.
- If **member1** is the root object, the **member1** object and its contents, and the **membership1** object are all exported as full objects.

POM revisioning

The revisable setting for business objects at both the source and destination systems must be consistent. Therefore, a class that has revisioning enabled at an exporting site has to have revisioning enabled at the

importing site; otherwise, the import fails. Only the latest version of a revisable object is transferred for both managed and unmanaged use cases.

Infer delete of 4GD objects

- **Mdl0SubsetDefinition**

  Deletes the search criteria if it is different from that in the imported TC XML file.

- **Cpd0DesignElementImpl**

  Causes a category change for these objects:

Original object	New object	Result
Shape design element	Reuse design element	Change the source object of the design element from an item revision to a shape revision.
Reuse design element	Shape design element	Change the source object of the design element from a shape revision to an item revision.
Reuse design element	Reuse design element	Deletes the subordinate design elements that are not in the imported TC XML file.

- **Workset**

  Deletes the **Subsets** that are not in the imported TC XML file.

  For the subassembly, infer delete is controlled by the **BVR** and **PSOccurrence** objects.

- **SubsetInstance in a Workset**

  Deletes the search criteria if it is different from that in the imported TC XML file.

  Deletes the realization maps that are not in the TC XML file.

- **Partition**

  Deletes the search criteria if it is different from that in the imported TC XML file.

  Deletes the child-parent links that not in the imported TC XML file.

  Deletes the memberships that are not in the imported TC XML file.

Configured export

1. Constructs the **BOMWindow** object for the given root object and configuration context (revision rule, variant rule, and effectivity).
2. Gets the top line from **BOMWindow** object. For a **Workset** object, this is the **WorksetLine** object.
3. Traverses from the top line as defined by the closure rules.
4. Serializes the traversed data with properties defined by the property set.

Exporting 4GD data to PLM XML

The **4GDPIEDataExportDefault** transfer mode is the default transfer mode for transferring 4GD data to PLM XML.

> Caution:
>
> Exporting 4GD (4th Generation Design) data in PLM XML format requires the out-of-the-box (OOTB) closure rules and property sets defined in the **4GDPIEDataExportDefault** transfer mode. If you need a custom transfer mode, Siemens Digital Industries Software recommends that you copy the OOTB transfer mode and add any required closure rules or property sets to the transfer mode. Do not delete any of the OOTB clauses, this can cause incorrect data in the exported PLM XML

Supported 4th Generation Design (4GD) root objects

Teamcenter supports PLM XML export for the following 4th Generation Design (4GD) root objects:

- **Cpd0DesignElement**
- **Cpd0DesignFeature**
- **Cpd0Workset**
- **Mdl0SubsetDefinition**
- **Ptn0Partition**
- **Partition**

Supported 4GD PLM XML export features

Supported	Unsupported
Property set customization	Filter rule customization
Closure rule customization	Action rule customization
Localization attribute export	Runtime property traversal in closure rule
Dataset/Form override	Dataset/form absolute override
Table property export	**AccesIndent/ownerRefs**

Supported	Unsupported
	LOV localization
	Delta export

Limitations exporting 4GD objects to PLM XML

4GD export to PLM XML has the following limitations:

- The configuration (revision rule, variant rule, effectivity, and so on) on a subset element is used when you export a workset.
- The configuration on the 4GD application view is ignored for the export.
- If some objects are not accessible, the corresponding elements and attributes may be missing in the PLM XML file. Missing objects or attributes may cause the schema validation to fail.
- If you set the **TIE_debug** value to **2**, no error occurs if some objects in the export are not accessible. The inaccessible objects are identified as configured out in the log file.
- The **4GDPIEDataExportDefault** transfer mode is supported only for 4GD exports. Do not use it for BOM exports, as this causes undefined behavior.
- 4GD subset items cannot be directly exported to 4GD PLM XML using the **4GDPIEDataExportDefault** transfer mode. To export subset items, either use the PLM XML **ConfiguredDataExportDefault** transfer mode instead of **4GDPIEDataExportDefault** or open the item in Structure Manager and export it as a BOM structure.
- The search option for a 4GD configuration is not supported.

Mapping 4GD object types to PLM XML

By default, design features and attribute groups are not exported with a workset structure because the **opt_4gd_exp_weld** and **opt_4gd_exp_ag** options are set to **FALSE** in the **TIEPLMXMLExportInternal** option set. If you want design features and attribute groups exported with the workset structure, set the **opt_4gd_exp_weld** and **opt_4gd_exp_ag** options to **TRUE**.

4GD object type	PLM XML element
ApprSearchCriteriaGroup	**SetFilter**
ApprSearchCriteriaBoxZone **ApprSearchCriteriaNamedZone** **ApprSearchCriteriaPlaneZone** **ApprSearchCriteriaProximity**	**ZoneFilter**
ApprSearchCriteriaSavedQry	**SavedQuery**

4GD object type	PLM XML element
Cpd0CollaborativeDesign	**CollaborativeDesign**
Cpd0Workset	**Workset**
Cpd0WorksetRevision	**WorksetRevision**
Cpd0WorksetMaster **Cpd0WorksetRevisionMaster** and other primary forms	**Form**
Cpd0DesignElement	**DesignElement**
Cpd0ContinuousJoin	**ContinuousJoinDesignFeature**
Cpd0ArcWeld	**ArcWeldDesignFeature**
Cpd0Datum	**DatumDesignFeature**
Cpd0DiscreteJoin	**DiscreteJoinDesignFeature**
Cpd0ResistanceWeld	**ResistanceWeldDesignFeature**
Cpd0SurfaceAdd	**SurfaceAddDesignFeature**
Cpd0AdhesiveFill	**AdhesiveFillDesignFeature**
Cpd0SurfaceWeld	**SurfaceWeldDesignFeature**
Mdl0ApprSrchCrtClosure	**ClosureRule**
Mdl0ApprSrchCrtOptionSet	**OptionSet**
Mdl0ApprSrchCrtType	**TypeSearchCriteria**
Mdl0ApprSrchGeomConstraint	**GeometricConstraint**
Mdl0ApprSrchSlctContent	**GroupFilter**
Mdl0AttributeGroup	**AttributeGroup**
Mdl0AttachAttributeGroup	**GeneralRelation**
Mdl0DefaultGeometry	**SubElement**s of **DesignElement**: **plm:Transform, plm:Bound** and **plm:Representation**
Mdl0ElementThread	**InstanceThread**
Cpd0ArcWeld	**ArcWeldDesignFeature**
Ptn0ApprSrchPartition	**PartitionSearchCriteria**
Ptn0Partition and its subtypes: **Ptn0Design**, **Ptn0Functional**, **Ptn0Mfg**, **Ptn0System, Ptn0Zone**	**Partition**
Ptn0PartitionItem and its subtypes: **Ptn0DesignItem**, **Ptn0FunctionalItem**, **Ptn0MfgItem**, **Ptn0SystemItem, Ptn0ZoneItem**	**PartitionItem**
Ptn0PartitionItemRevision and its subtypes: **Ptn0DesignItemRevision**,	**PartitionItemRevision**

4GD object type	PLM XML element
Ptn0FunctionalItemRevision, **Ptn0MfgItemRevision**, **Ptn0SystemItemRevision**, **Ptn0ZoneItemRevision**	
Ptn0PartitionScheme	**PartitionScheme**

Importing 4GD PLM XML

You can update 4GD data by importing 4GD PLM XML created or modified by third-party applications. Contact your Siemens Digital Industries Software representative for more information on working with 4GD data.

Imported objects are merged with your existing 4GD data. By default, if an imported object already exists, the existing 4GD object is updated to match the imported object. Importing 4GD PLM XML requires the **4GDPIEDataImportDefault** transfer mode delivered with Teamcenter.

Imported 4GD PLM XML must adhere to the PLM XML standard and its restrictions. A 4GD PLM XML file is identified to Teamcenter and differentiated from a standard PLM XML file by its **Header** element's child **Application** element having its name attribute set to **4G**. For example:

```
<Header id="id1" traverseRootRefs="#id2"
transferContext="4GDPIEDataExportDefault">
    <Application id="id19" name="4G" version="0"/>
</Header>
```

If you are exchanging 4GD data between Teamcenter sites, export and import data using TC XML. If you are importing BOM data that does not include 4GD data, import using standard PLM XML.

4GD PLM XML limitations

Be aware of the following limitations when importing 4GD PLM XML:

- Importing the **Representation** 4GD PLM XML element (a child element of **Occurrence**) is not supported.
- Importing 4GD PLM XML as **PSOccurrenceNotes** and **BoundingMultiBox** objects is not supported.
- **Add** and **Modify** elements (used in delta PLM XML) are not imported in 4GD PLM XML.
- The PLM XML schema allows an element in one file to reference another PLM XML file. Such references are not supported when importing 4GD PLM XML files. In these cases, import the XML files separately.
- Importing closure rules is not supported. All the data in a 4GD PLM XML file is imported and is not filtered by import closure rules.

- Only major revisions are imported. Change space and minor revisions are not supported for import.
- Effectivity and variant configuration information is not imported.
- Run-time properties are not imported. Persistent attributes in **UserData** are imported.
- Business Modeler IDE extensions for operations on business objects (such as creating and saving) are not honored or executed when importing.
- Revising objects during importing (that is, importing a new revision of an existing object and copying or linking the related data from the old revision to the new revision) is not supported.
- 4GD override data and constraints data are not supported.
- 4GD connection groups and links are not supported.
- Changing a design element category during importing is not supported.
- Importing BOM incremental change data is not supported.

Mapping 4GD PLM XML to 4GD object types

You can update your 4GD data by importing 4GD PLM XML created or modified by third-party applications.

4GD PLM XML-to-object import mapping

The **4GDPIEDataImportDefault** transfer mode is the default transfer mode for importing 4GD PLM XML. The following 4GD PLM XML elements are mapped to the listed 4GD object types on import.

4GD PLM XML element	4GD object type
AdhesiveFillDesignFeature	**Cpd0AdhesiveFill**
ArcWeldDesignFeature	**Cpd0ArcWeld**
ArcWeldDesignFeature	**Cpd0ArcWeld**
AttributeGroup	**Mdl0AttributeGroup**
ClosureRule	**Mdl0ApprSrchCrtClosure**
CollaborativeDesign	**Cpd0CollaborativeDesign**
ContinuousJoinDesignFeature	**Cpd0ContinuousJoin**
DatumDesignFeature	**Cpd0Datum** and its source object **Cpd0ShapeDesign**
DesignElement	**Cpd0DesignElement** **Cpd0DesignItemInstance** **Rlz0ItemRealizationMap**

4GD PLM XML element	4GD object type
DesignElement subelements **Transform**. **Bound**, **Representation**	**Mdl0DefaultGeometry**
DesignFeatureBase and its subelements **Transform**, **Bound**, **Representation**	**Cpd0DesignFeature** **Mdl0DefaultGeometry**
DiscreteJoinDesignFeature	**Cpd0DiscreteJoin**
Form	**Cpd0WorksetMaster**, **Cpd0WorksetRevisionMaster**, and other primary forms
GeneralRelation	**Mdl0AttachAttributeGroup**
GeometricConstraint	**Mdl0ApprSrchGeomConstraint**
GroupFilter	**Mdl0ApprSrchSlctContent**
InstanceThread	**Mdl0ElementThread**
Member	**Ptn0Membership**
OccurrenceFilter	**ApprSearchCriteria**
OptionSet	**Mdl0ApprSrchCrtOptionSet**
Partition	**Ptn0Partition** and its subtypes **Ptn0Design**, **Ptn0Functional**, **Ptn0Mfg**, **Ptn0System**, **Ptn0Zone**, **Ptn0ChildParentLink**
PartitionItem	**Ptn0PartitionItem** and its subtypes **Ptn0DesignItem**, **Ptn0FunctionalItem**, **Ptn0MfgItem**, **Ptn0SystemItem**, **Ptn0ZoneItem**
PartitionItemRevision	**Ptn0PartitionItemRevision** and its subtypes **Ptn0DesignItemRevision**, **Ptn0FunctionalItemRevision**, **Ptn0MfgItemRevision**, **Ptn0SystemItemRevision**, **Ptn0ZoneItemRevision**
PartitionScheme	**Ptn0PartitionScheme**
PartitionSearchCriteria	**Ptn0ApprSrchPartition**
Product	**Cpd0ShapeDesign**
Product	**Item**
ProductRevision	**Cpd0ShapeDesignRevision**
ResistanceWeldDesignFeature	**Cpd0ResistanceWeld**
SavedQuery	**ApprSearchCriteriaSavedQry**
SetFilter	**ApprSearchCriteriaGroup**
Subset	**Cpd0DesignSubsetElement/Mdl0SubsetDefinition**
SubsetInstance	**Cpd0DesignSubsetInstance**
SurfaceAddDesignFeature	**Cpd0SurfaceAdd**
SurfaceWeldDesignFeature	**Cpd0SurfaceWeld**

4GD PLM XML element	4GD object type
TypeSearchCriteria	Mdl0ApprSrchCrtType
Workset	Cpd0Workset
WorksetRevision	Cpd0WorksetRevision Cpd0WorksetModel
ZoneFilter	ApprSearchCriteriaBoxZone ApprSearchCriteriaNamedZone ApprSearchCriteriaPlaneZone ApprSearchCriteriaProximity

The following PLM XML elements are mapped to the listed Teamcenter object types.

4GD PLM XML element	4GD object type
ApplicationRef	PLMAppUID
AssociatedAttachment	GRM
AssociatedDataSet	ImanRelation
AssociatedForm	ImanRelation
DataSet	Dataset
Form	Form
Occurrence	BOMLine
Product	Item
ProductInstance	PSOccurrence
ProductRevision	ItemRevision

Exporting data to Supplier Relationship Management (SRM)

Configure data export to Supplier Relationship Management

To export Teamcenter data to Supplier Relationship Management (SRM), you must set the host and port where SRM Connect is running in the **SRM_hostname_portnumber** preference.

SRM Connect can be integrated to Teamcenter using a socket service connection.

For information about SRM Connect, see the *SRM Connect Reference Guide*.

By default, SRM export uses the **TIEUnconfiguredExportDefault** transfer option set (TOS). You can create a custom transfer option set for your SRM export using the PLM XML/TC XML Export Import Administration application.

Add the following properties to the **TIEExportDefaultPS** property set for the **TIEUnconfiguredExportDefault** TOS:

Primary object class type	Primary object	Relation type	Related property or object	Property action type
CLASS	Item	Attribute	checked_out	DO
CLASS	Item	Attribute	checked_out_user	DO
CLASS	Item	Attribute	object_type	DO
CLASS	ItemRevision	Attribute	checked_out	DO
CLASS	ItemRevision	Attribute	checked_out_user	DO
CLASS	ItemRevision	Attribute	object_type	DO
CLASS	Dataset	Attribute	object_type	DO

You can provide a list of values (LOV) for the Partition and Template for a bid package that allows users to select what is appropriate for their package. These Partition and Template LOVs must be present in Teamcenter before they can be used to export data to SRM.

Also, when you export an SRM object that has custom item or subitem types, the item type is exported as an attribute on a standard Teamcenter item object parent for the item. This approach maintains object information during transfers between systems. To support this feature, you must update the RFx maintained properties and XML file to store the types and subtypes supported by Teamcenter.

Export Teamcenter data to SRM

Before you can export objects to Supplier Relationship Management (SRM), **configure SRM data sharing**.

1. In the rich client, select the part or assembly object that you want to export.
2. Choose **Tools→Export→To SRM** to open the **Export To SRM** dialog box.
3. Enter a **Sponsor Email** value. The default email address is your Teamcenter user email address. The exported data will be emailed to this address.
4. Select a partition from the **SRM Partition** list.
5. Select a template from the **SRM Template** list.
6. (Optional) Type a description in **Reason** for future reference.
7. (Optional) Select a transfer option set from the **Option Set** list.

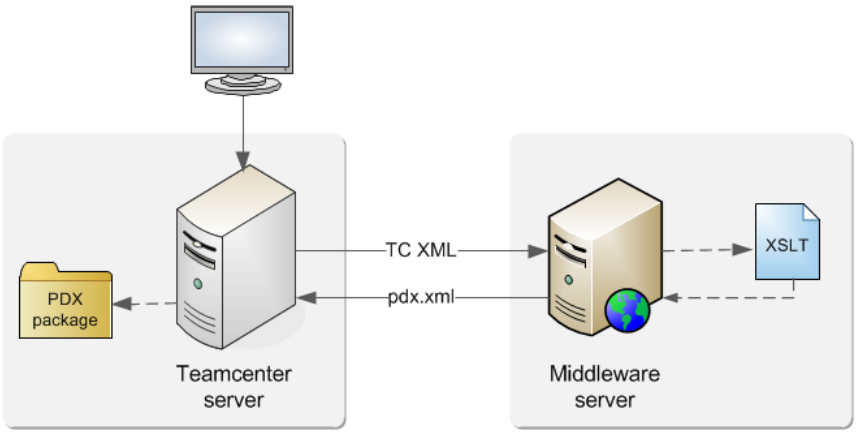
8. Click **OK** to start the export and close the dialog box. Teamcenter sends an email to the **Sponsor Email** with the exported data, along with whatever other information is specified by the transfer option set.

Exporting data to PDX

PDX export

Data exports in PDX format as a PDX package. This package can be viewed by PDX viewer applications or used by other systems that support the PDX data format. You can also create a PDX package using a workflow process by specifying the workflow handler provided for PDX packages.

Teamcenter data is exported in an XML format (TC XML) that is converted to PDX format by a style sheet (XSLT). Teamcenter uses features of the Teamcenter integration framework and Teamcenter briefcase to perform the export. Teamcenter schedules the export job and also transforms the TC XML data to PDX data using XSLT. Teamcenter briefcase packages the contents and uploads the PDX package to Teamcenter. The following figure shows how the Teamcenter components interact to export a PDX package.



The site definition distinguishes a PDX export from a standard briefcase export. When you define a site as a target for PDX export, you must identify the following items as described in **Configure PDX export and define a PDX workflow**:

- The integration framework deployment URL.
- The style sheet that is applied to this TC XML file in the data mapping step in integration framework.

When exporting the TC XML data, you must use a transfer option set that has its **opt_pdx_export** option set to a value of **True**.

Teamcenter objects that can be exported to PDX and their default mapping are described in *Default mapping of Teamcenter objects to PDX objects*.

Configure PDX export and define a PDX workflow

Prerequisites

Caution:

You must select the **Vendor Management** feature during Teamcenter installation to create the **TIEPDXPropertySet** property set that PDX export requires.

Teamcenter Environment Manager (TEM) imports the *defaultTransfermodes.xml* file that contains the definitions of all standard transfer modes. The importer validates the clauses of a property set. Because the **TIEPDXPropertySet** property set contains vendor management-related clauses, if you do not select the **Vendor Management** feature, the validation fails when the vendor management-related clauses are encountered. This prevents the PDX export feature from being fully functional.

Configuring PDX export

1. Map your Teamcenter objects and their attributes to PDX objects and their attributes. Teamcenter provides a default mapping of standard Teamcenter objects to PDX objects (see *Default mapping of Teamcenter objects to PDX objects*) and a default style sheet for transforming standard objects to PDX objects.

    a. Map any custom subtypes or data model changes.

    b. If the default PDX transfer option set and transfer mode are not appropriate for mapping your PDX package due to new subtypes or data model changes, define an appropriate transfer option set and transfer mode.

        A. Copy the default transfer mode file. Siemens Digital Industries Software recommends that you name the transfer mode file based on the business process.

        B. Update the transfer mode to reference the information that must be exported in a PDX package. For details on how to change the closure rule, filter rule, property set, and revision rule, see *Tc XML and PLM XML Configuration for Data Import and Export*.

        C. Set the **opt_pdx_export** option to a value of **True**. Only those option sets with this option set to **True** are displayed in the **PDX Export** dialog box. (Option sets delivered with Teamcenter have this option set to **True** by default.)

    c. Create a custom style sheet to transform the custom subtypes or data model differences.

        A. Copy the standard PDX style sheet (*Tc2PDX.xsl* file) located in the *TC_DATA\data* directory of your Teamcenter installation.

            Siemens Digital Industries Software recommends renaming the style sheet, for example, *WidgetCorpTc2PDX.xsl*.

B. Change the style sheet to transform the information required in your exported PDX package.

C. Ensure this style sheet is called during export of a PDX package.

2. Define the PDX remote site:

a. Select the top-level sites node  from the **Organization List** tree.

b. In the **Sites** pane, enter a descriptive name for the site in **Site Name**, for example, PDX site 1: **PDX1**.

c. Enter a unique identifier in **Site ID**, for example, **1009598626**.

Each site must be defined at other sites using exactly the same site ID in each definition.

d. Select **Uses TCXML Payload**.

TcGS URL is read only until you select **Uses TCXML Payload**.

e. For Teamcenter Integration Framework based export, set **TcGS URL** to the URL used to contact the Teamcenter Integration Framework web server port, for example:

```
http://hostname:8080/tcif
```

f. Select **Is Offline**.

g. Click **Create**.

3. Attach the customized *Tc2PDX.xsl* style sheet to the default PDX **TIEPDXExportDefault** transfer mode:

a. In a Teamcenter command shell, use the **plmxml_tm_edit_xsl** utility to attach the style sheet to the transfer mode, for example:

```
plmxml_tm_edit_xsl -u=tc-admin-user -p=password
-transfermode=TIEPDXExportDefault
-xsl_file=C:\TC\tcdata\data\WidgetCorpTc2PDX.xsl
-action=attach
```

b. Use the utility to verify the style sheet is attached correctly, for example:

```
plmxml_tm_edit_xsl -u=tc-admin-user -p=password
-transfermode=TIEPDXExportDefault -action=list
```

The utility's output should be similar to:

```
Original File Name is <Tc2PDX.xsl>
```

4. Set the **PDX_pkg_file_name** preference at the site location. The following keywords are the only keywords supported by **PDX_pkg_file_name**. The keywords are case sensitive and other values are treated as constants.

PDX-string	*PDX-string* is a string constant prefixed to the package file name of each exported package. You can use this to identify the package as desired, such as by the originating site.
ItemId	Includes the item ID of the exported object as part of the package file name.
ItemName	Includes the item name of the exported object as part of the package file name.
RevId	Includes the revision ID of the exported object as part of the package file name.
TargetSite	Includes the site name the object exported to as part of the package file name.
TimeStamp	Includes the time stamp information at the time the object is exported as part of the package file name.

5. Set the **PROCESS_WARM** and **PROCESS_TARGET** parameter in the server pool properties file (*serverPoolconfiguration-ID.properties*) to ensure there is a sufficient number of tcservers available. The following are recommended minimum values for PDX export functionality:

```
PROCESS_WARM=5
PROCESS_TARGET=0700 10, 1700 10
```

6. Test the customized solution.

7. Provide your users with the transfer option set name they must select when they are exporting a PDX package.

Defining a PDX workflow

The **BC-perform-offline-export** workflow handler contains the required and optional arguments that, as an administrator, you can use to create workflow process templates for exporting a PDX package through a workflow.

Creating a custom PDX style sheet

Mapping subtyped objects to PDX elements

The standard style sheet (*Tc2PDX.xsl*) maps elements of TC XML data to elements of PDX data as follows.

TC XML element	TC XML element
Item, Part, CommercialPart	**Item**
EngChange	**Change**

TC XML element	TC XML element
ManufacturerPart	ManufacturerPart
Vendor	Contact

If these Teamcenter objects are subtyped, their corresponding sections have to be duplicated as shown in the following *Customized style sheet with subtypes* example. The standard style sheet makes extensive use of templates (to generate **BOMLine** elements, **Attachment** elements, and so forth). Your customizations must preserve their functionality.

Collapsed standard style sheet view

```
<xsl:if test=&#8221;count(//tc:Item)&gt;0
         or count(//tc:Part))&gt;0
         or count(//tc:CommercialPart))&gt;0&#8221;>
    <xsl:element name=&#8221;Items&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:Item&#8221;>
            &#8942;
        <xsl:for-each select=&#8221;tc:TcXML/tc:Part&#8221;>
            &#8942;
        <xsl:for-each select=&#8221;tc:TcXML/tc:CommercialPart&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>

<xsl:if test=&#8221;count(//tc:EngChange)&gt;0&#8221;>
    <xsl:element name=&#8221;Changes&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:EngChange&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>

<xsl:if test=&#8221;count(//tc:ManufacturerPart)&gt;0&#8221;>
    <xsl:element name=&#8221;ManufacturerPart&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:ManufacturerPart&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>

<xsl:element name=&#8221;History&#8221;>
    &#8942;
</xsl:element>

<xsl:if test=&#8221;count(//tc:Vendor)&gt;0&#8221;>
    <xsl:element name=&#8221;Contacts&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:Vendor&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>
```

Collapsed standard style sheet view

```
<xsl:if test=&#8221;count(//tc:Item)&gt;0
         or count(//tc:Part))&gt;0
         or count(//tc:MyCustomPart))&gt;0
```

```
         or count(//tc:CommercialPart))&gt;0&#8221;>
    <xsl:element name=&#8221;Items&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:Item&#8221;>
            &#8942;
        <xsl:for-each select=&#8221;tc:TcXML/tc:Part&#8221;>
            &#8942;
        <xsl:for-each select=&#8221;tc:TcXML/tc:MyCustomPart&#8221;>
            &#8942;
        <xsl:for-each select=&#8221;tc:TcXML/tc:CommercialPart&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>

<xsl:if test=&#8221;count(//tc:EngChange)&gt;0&#8221;>
    <xsl:element name=&#8221;Changes&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:EngChange&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>

<xsl:if test=&#8221;count(//tc:ManufacturerPart)&gt;0&#8221;>
    <xsl:element name=&#8221;ManufacturerPart&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:ManufacturerPart&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>

<xsl:element name=&#8221;History&#8221;>
    &#8942;
</xsl:element>

<xsl:if test=&#8221;count(//tc:Vendor)&gt;0&#8221;>
    <xsl:element name=&#8221;Contacts&#8221;>
        <xsl:for-each select=&#8221;tc:TcXML/tc:Vendor&#8221;>
            &#8942;
    </xsl:element>
</xsl:if>
```

Mapping custom attributes to PDX attributes and elements

Custom attributes of Teamcenter objects must be mapped either to attributes of PDX elements or to **AdditionalAttributes** PDX elements. Add these tags as shown in the following example for the **attachment** element.

Customized style sheet with AdditionalAttributes elements

```
<xsl:attribute name="fileSize&#8942;>
    <xsl:value&#8212;of
        select=&#8942;//tc:ImanFile/tc:GSIdentity(@elemId=$dsId)/../@byte_size"/>
</xsl:attribute>
<xsl:attribute name="versionIdentifier&#8942;>
    <xsl:value&#8212;of
        select=&#8942;//tc:GSIdentity(@elemId=$imsoId)/../@version_number"/>
</xsl:attribute>
<xsl:attribute name="attachmentModificationDate&#8942;>
    <xsl:value&#8212;of
        select=&#8942;//tc:ImanFile/tc:GSIdentity(@elemId=$dsId)/../@last_mod_date"/>
```

```
</xsl:attribute>
<xsl:attribute name="extraAttribute&#8942;>
    <xsl:value&#8212;of select=&#8942;Yes"/>
</xsl:attribute>
<xsl:element name="AdditionalAttributes&#8942;>
    <xsl:element name="AdditionalAttribute&#8942;>
        <xsl:attribute name="name&#8942;>
            <xsl:text>CustomAttribute1</xsl:text>
        </xsl:attribute>
        <xsl:attribute name="value&#8942;>
            <xsl:value&#8212;of
             select=&#8942;//tc:ImanFile/tc:GSIdentity(@elemId=$dsId)/../
@myCustomAttribute"/>
        </xsl:attribute>
    </xsl:element>
</xsl:element>
```

PDX elements supported for export by default

The standard style sheet (*Tc2PDX.xsl*) transforms a subset of PDX elements. The style sheet must be extended for any additional PDX specification elements you want to support (for example, **SupplierPart**). Use the standard style sheet as a reference for adding the additional elements.

The following PDX elements are supported by default:

- **AdditionalAttribute**
- **AffectedItem**
- **ApprovedManufacturerList**
- **ApprovedManufacturerListItem**
- **Attachment**
- **BillOfMaterial**
- **BillOfMaterialItem**
- **BillOfMaterialMarkup**
- **Change**
- **Contact**
- **HistoryItem**
- **Item**
- **ManufacturerPart**
- **ProductDataeXchangePackage**
- **ReferenceDesignator**

Exporting PDX packages

Working with PDX packages

A PDX package is a dataset that is accessible from a workflow process like other datasets. You can attach or refer a PDX package to another object in a workflow process. You can copy a document to be sent as a reference document in the process dialog box.

Use the following steps to query for a PDX package.

1. Click **Open Search View** on the rich client toolbar.

2. In the **Search** pane, click and select **More** to open the **Change Search** dialog box.

3. In the **System Defined Searches** list, select **Dataset** and click **OK**.

4. In the **Search** pane, set **Dataset Type** to **PDX** and click to perform the search.

Export PDX packages

The PDX export feature is available within the rich client from the **Tools** menu. PDX export can be used within a workflow and a PDX package can be accessed within a workflow the same as any dataset.

The Teamcenter objects that can be exported to PDX and their default mapping are described in *Default mapping of Teamcenter objects to PDX objects*.

1. Select the object to export.

2. Choose **Tools→Export→To PDX**. Unmanaged sites are not listed as available export sites for PDX.

3. In the **Export to PDX** dialog box:

   a. (Optional) Enter the reason you are exporting the package in **Reason**.

   b. Click **Select target remote site(s)** next to **Target Sites** to select the export destinations for the objects.

   c. Select the PDX option set from the **Option Set** list.

   d. (Optional) Select your desired options:

Start immediate transfer	Starts the export process immediately instead of during the scheduled off-hours period.
Show progress indicator	Displays the progress pane automatically when the process starts.
Send email notification	Send an email when the transfer is complete to the address associated with the requesting user.
Delta Export	Export only structures modified since the last export.

   e. (Optional) Click **Display/Set export options** to select the transfer options for the PDX package.

The options available in this dialog box depend on the variables in the selected option set.

4. Click **Yes** to export the data.

   After the export completes, open your **Mailbox** to locate the PDX package. You can copy the package to another location in Teamcenter or download the package to the file system.

Download a PDX package

1. Right-click on the PDX package and choose **Named References** to display the **Named References** dialog box.

2. Select the PDX package file in the **Name** column and click **Export**.

3. Enter the location where you want to place the PDX package and click **Export**.

Default mapping of Teamcenter objects to PDX objects

By default, the following set of Teamcenter objects are mapped to the listed PDX objects.

PDX element name	PDX attribute	Teamcenter object	Teamcenter attribute
ApprovedManufacturerListItem	manufacturerPartIdentifier	ManufacturerPart	object_name
ApprovedManufacturerListItem	manufacturerPartUniqueIdentifier	ManufacturerPart	item_id
ApprovedManufacturerListItem	manufacturerContactUniqueIdentifier	Vendor	item_id
ApprovedManufacturerListItem	globalManufacturerPartStatusCode	ManufacturerPart Revision →ReleaseStatus	name
ApprovedManufacturerListItem	globalManufacturerPartStatusCodeOther	ManufacturerPart Revision →ReleaseStatus	name
ApprovedManufacturerListItem	globalPreferredStatusCode	VendorIdentifier	preferred_status
ApprovedManufacturerListItem	description	ManufacturerPart	object_desc
ApprovedManufacturerListItem	manufacturedBy	VendorIdentifier	object_name
AffectedItem	itemIdentifier	Item	object_name
AffectedItem	itemUniqueIdentifier	Item	item_id
AffectedItem	oldRevision	ItemRevision	item_revision_id
AffectedItem	newRevision	ItemRevision	item_revision_id
AffectedItem	obsoleteDate	ItemRevision →ReleaseStatus	date_released
AffectedItem	effectiveDate	ItemRevision →ReleaseStatus	date_released
AffectedItem	globalLifeCyclePhaseCode	ItemRevision →ReleaseStatus	name

PDX element name	PDX attribute	Teamcenter object	Teamcenter attribute
AffectedItem	globalLifeCyclePhaseCodeOther	ItemRevision →ReleaseStatus	name
AffectedItem	description	ItemRevision	object_desc
Attachment	universalResourceIdentifier	Imanfile	original_file_name
Attachment	fileIdentifier	Imanfile	file_name
Attachment	fileSize	Imanfile	byte_size
Attachment	description	Dataset	description
BillOfMaterialItem	revisionIdentifier	PSOccurrence Item Item Revision	item_revision_id
BillOfMaterialItem	globalBillOfMaterialTypeCode	Item	object_type
BillOfMaterialItem	globalBillOfMaterialTypeCodeOther	Item	object_type
BillOfMaterialItem	notes	PSOccurrence →PSOccurrenceNotes	note_texts
BillOfMaterialItem	BillOfMaterialItemIdentifier	PSOccurrence →Item	object_name
BillOfMaterialItem	billOfMaterialItemUniqueIdentifier	PSOccurrence →Item	item_id
BillOfMaterialItem	itemQuantity	PSOccurrence	qty_value
BillOfMaterialItem	description	PSOccurrence →Item →Item Revision	object_desc
BillOfMaterialItem	proprietarySequenceIdentifier	PSOccurrence	seq_no
Change	changeNumber	EngChange	item_id
Change	revisionIdentifier	EngChange Revision	item_revision_id
Change	globalEngineeringChangeStatusCode	EngChange Revision →ReleaseStatus	name
Change	globalEngineeringChangeStatusCodeOther	EngChange Revision →ReleaseStatus	name
Change	changeStatusDate	EngChange Revision →ReleaseStatus	date_released
Change	changeType	EngChange	object_type
Change	changeSubType	EngChange Revision Master	ec_type
Change	changeOriginationDate	EngChange Revision	creation_date
Change	changeRequestDescription	CMII CR/CN Form	object_desc
Change	changeOwnerName	EngChange	owning_user
Change	description	EngChange Revision	object_desc
Contact	contactIdentifier	Vendor	object_name
Contact	contactUniqueIdentifier	Vendor	item_id
Contact	contactName	Vendor	contact_name
Contact	addressLine1	Vendor	supplier_addr

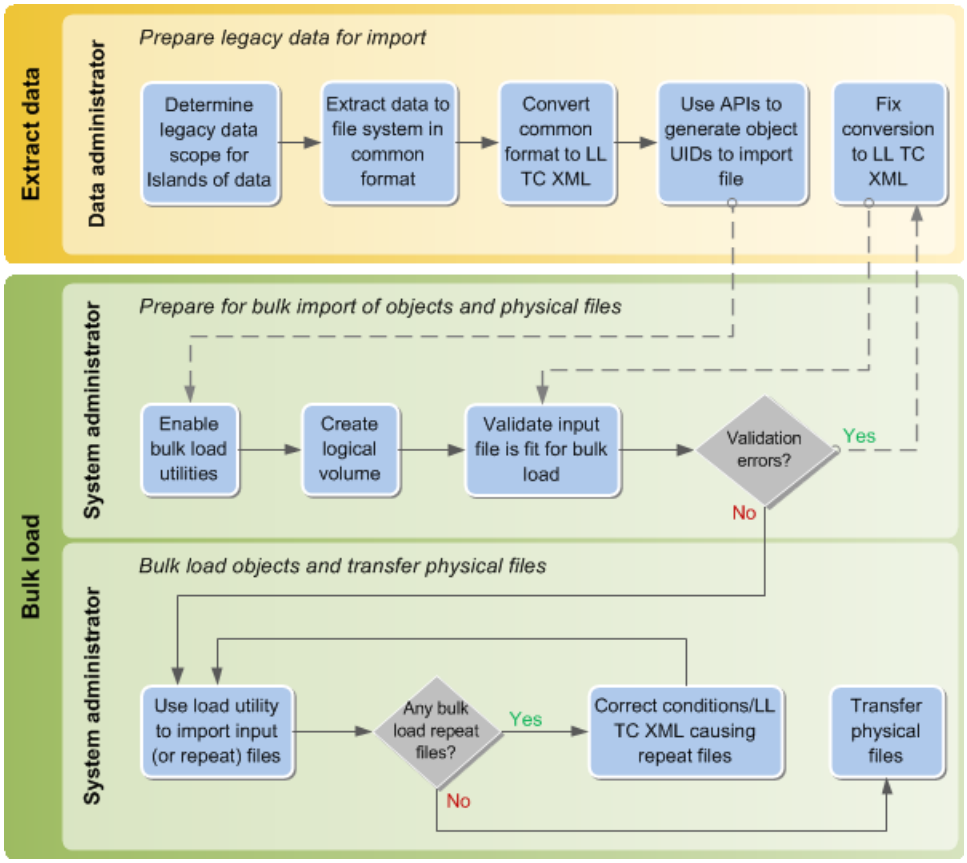
PDX element name	PDX attribute	Teamcenter object	Teamcenter attribute
Contact	telephoneNumber	Vendor	supplier_phone
Contact	emailAddress	Vendor	email_addr
HistoryItem	userName	Header	author
Item	itemIdentifier	Item	object_name
Item	itemUniqueIdentifier	Item	Item_id
Item	globalLlifeCyclePhaseCode	Item Revision →Release Status	name
Item	globalLlifeCyclePhaseCodeOther	Item Revision →Release Status	name
Item	globalProductTypeCode	Item	object_type
Item	revisionIdentifier	Item Revision	item_revision_id
Item	globalProductUnitOfMeasureCode	Item →UnitOfMeasure	unit
Item	revisionReleasedDate	Item Revision	release_date
Item	description	Item	object_desc
ManufacturerPart	manufacturerPartIdentifier	ManufacturerPart	object_name
ManufacturerPart	manufacturerPartUniqueIdentifier	ManufacturerPart	item_id
ManufacturerPart	manufacturerName	Vendor	object_name
ManufacturerPart	manufacturerContactUniqueIdentifier	Vendor	item_id
ManufacturerPart	globalManufacturerPartStatusCode	Manufacturer Part Revision →ReleaseStatus	name
ManufacturerPart	globalManufacturerPartStatusCodeOther	Manufacturer Part Revision → ReleaseStatus	name
ManufacturerPart	manufacturerPartType	ManufacturerPart	object_type
ManufacturerPart	description	ManufacturerPart	object_desc
ProductDataeXchangePackage	originatedByContactName	Header	author
ProductDataeXchangePackage	originatedByContactUniqueIdentifier	Header	author
ReferenceDesignator	referenceDesignatorName	PSOccurrence	ref_designator

3. Loading bulk data into Teamcenter from other sources

Understanding the process of loading bulk data

When migrating from Teamcenter Enterprise or when working with other third-party applications, you made need to import large amounts of data (bulk data) into your Teamcenter environment. Teamcenter's bulk load utilities use TC XML files that bypass the business logic rules when importing data. This approach provides performance enhancements when loading large amounts of data.

Acquiring and loading bulk data into your Teamcenter environment requires the following steps and roles:

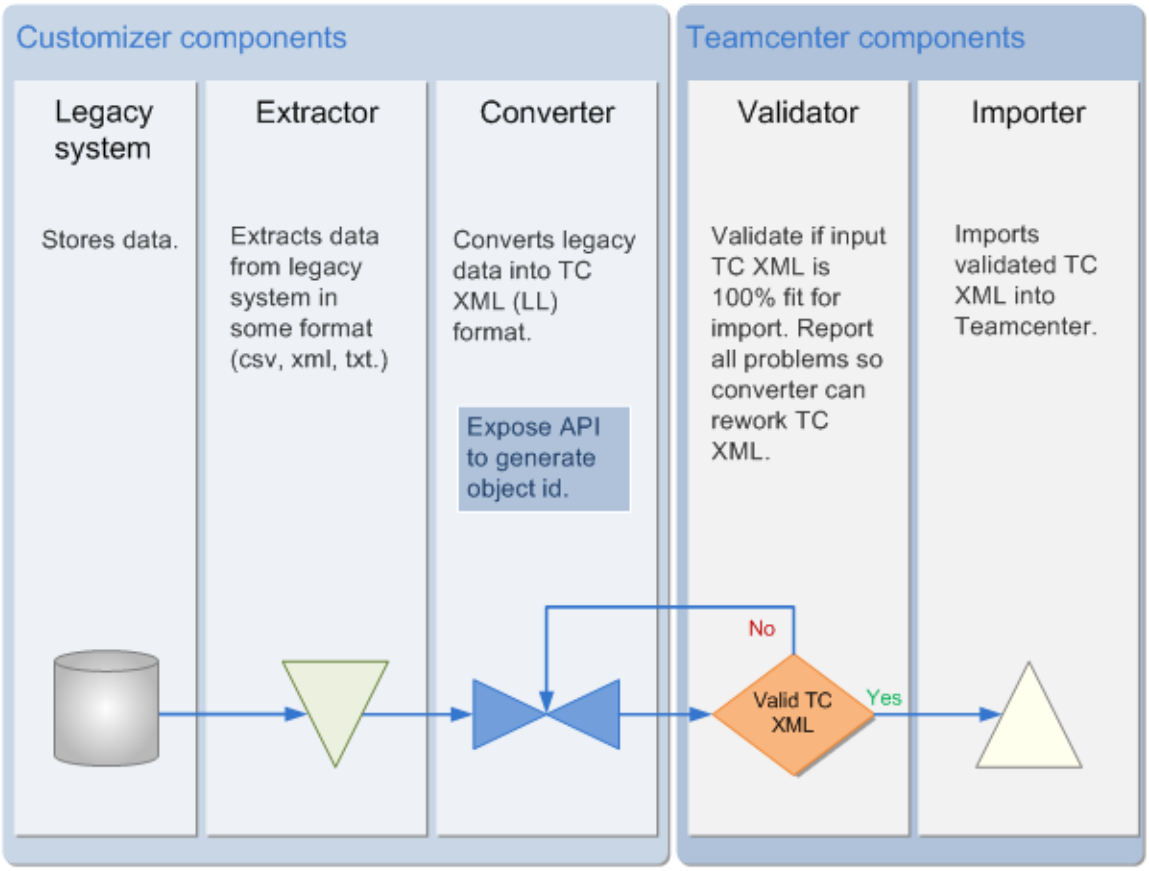


Business process	Key steps	Performed by
Extract legacy data for import	Determine the legacy data to extract into low-level TC XML files containing islands of data that the bulk load utility can use as import files.	Data administrator Implementation engineer Technical architect Siemens Digital Industries Software Services can assist

Business process	Key steps	Performed by
	Extract the legacy data to the common, comma-separated values (CSV), format that can easily be transformed to the required format. **Use the CSV to TC XML converter** to create the required TC XML import files.	in the conversion process using tools they have developed for this process.
Prepare Teamcenter, legacy data, and the associated physical files for import into Teamcenter	Obtain and create a bulk import **utility key** and configure a logical volume for physical files in Teamcenter.	System administrator
Validate the import file	Validate your TC XML import file meets Teamcenter requirements. Correct any validation errors.	System administrator
Bulk load objects	Load data and correct any issues that occur using **bulk load repeat** files.	System administrator

Bulk loading data

Bulk loader is a tool that you can use in a data migration solution to import data from a legacy system into Teamcenter. The bulk load business process leverages the site consolidation technology to achieve desired performance/scalability specifications for data migration. The legacy data must be extracted and mapped to the TC XML format by a custom solution. The following figure shows the components involved in a data migration solution. The Validator, Importer, and the API to generate object ID components are provided as part of the bulk loader functionality.



Bulk loader components

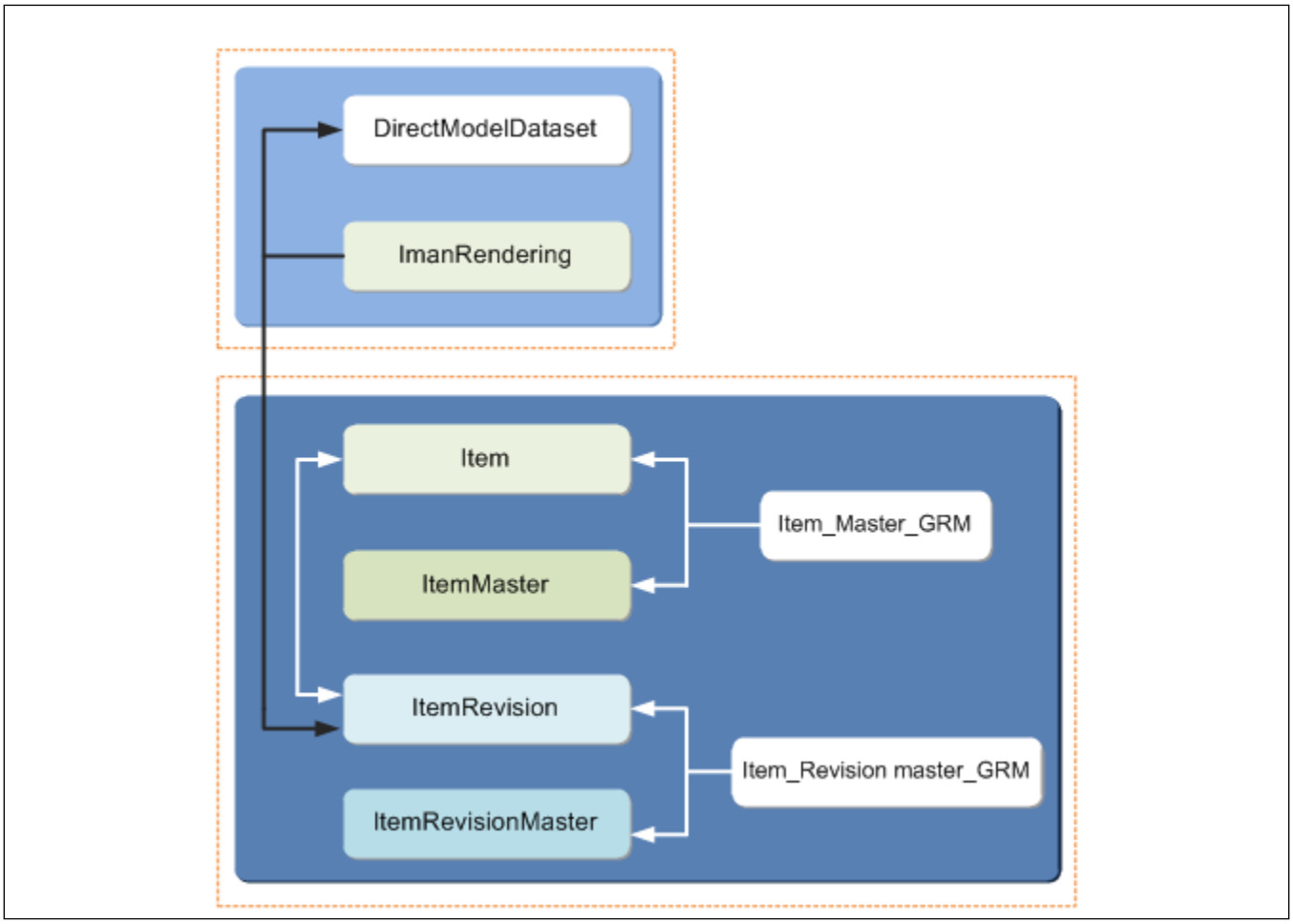
Working with bulk data from Teamcenter Enterprise or other systems

Understanding bulk data

Bulk data is loaded into Teamcenter using the concept of *islands of data*. Loading data in islands maintains the integrity of data relationships and allows recovery from any import failures during the bulk load. The following terms relate to this concept:

Island	The fundamental unit of data. It consists of a primary object and the additional objects on which it depends for its correct functional definition and usage within Teamcenter.
Primary object	The initial object used to start the traversal to determine an island of data.
Dependent object	An object that must be included in the set of objects implied by the primary object and the closure rules to correctly form the island of data. Dependent objects may not exist in a particular island.

The following diagram shows how an island can be defined:



Island of data

Understanding how TC XML applies to bulk data

Teamcenter's bulk data utilities work with **TC XML data**. The TC XML format is defined by the TC XML schema. The TC XML schema is autogenerated at installation and upgrade, and the resulting **TCXML.xsd** file is placed in the *TC_DATA* directory on the Teamcenter server. You can also use the Business Modeler IDE to export a TC XML schema file of your current Teamcenter schema.

In a TC XML file, the root element is **TCXML** and the namespace (**xmlns**) attribute is **http://www.tcxml.org/Schemas/TCXMLSchema**, for example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<TCXML xmlns="http://www.tcxml.org/Schemas/TCXMLSchema" format="low level" >
```

The TC XML root element has a format attribute that you can set to **low_level** or **high_level**. For bulk loading, you must set the attribute to **low_level**.

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

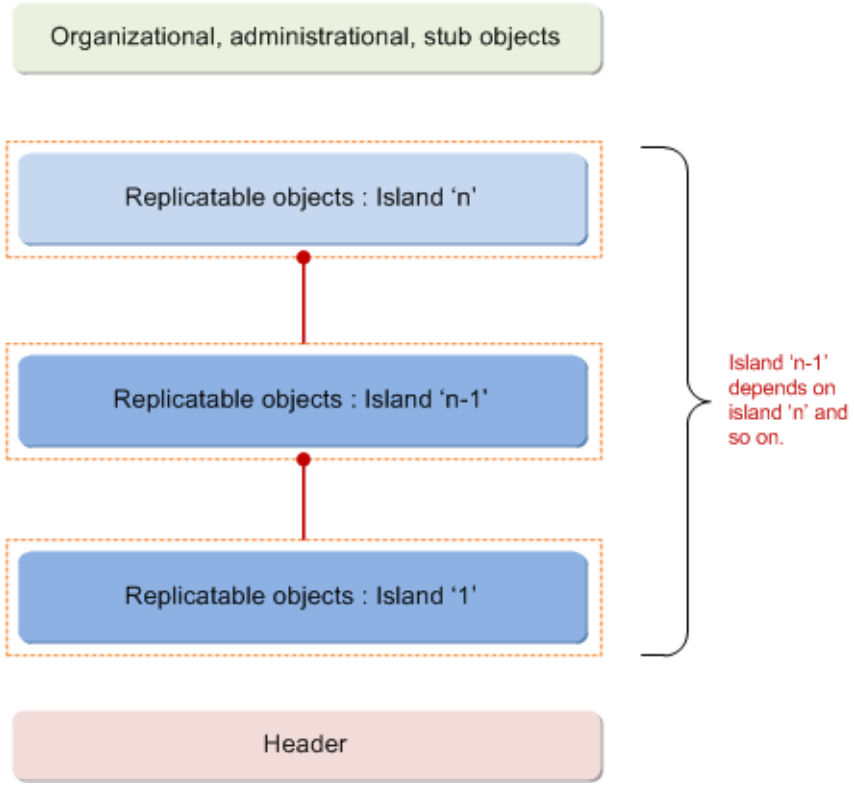
A TC XML file has a flat structure (no hierarchy). Elements in the TC XML file have:

- The leaf - class name as its element name.
- An **elemId** attribute to uniquely identify the element (required).
- An **island_id** attribute (required).

> Note:
>
> The **elemId** attribute value must have **id** followed by a number, for example, **elemId = "id90"**.

Object references in a TCXML file are formed by prefixing the hash character (**#**) to the **elemId** value as follows:

TC XML data is grouped into islands; an island ties logically related objects together. The data migration solution can form simple islands, like [**Item**, **ItemRev**, **MasterForm**] or [**Dataset**, **RevAnchor**, **ImanFile**], and the fast import processes the islands one - at - a - time. Use the following TC XML data island structure for the best results:



XML elements for organizational objects (**User**, **Group**, **Site**, and so forth) and administrative objects (**Types**, **Rules**) are assigned an **island_id** attribute value of **0** and contain the candidate key attribute.

The organizational and administrative objects are not replicated; they are looked up at the target site based on their candidate key.

```
<Group elemId="id1" island_id="0" name="dba"/>
<PSViewType elemId="id2" island_id="0" name="view"/>
<ImanType elemId="id3" island_id="0" type_class="ImanRelation" type_name="IMAN_specification"/>
<ImanType elemId="id4" island_id="0" type_class="ImanRelation" type_name="IMAN_master_form"/>
<User elemId="id5" island_id="0" user_id="infodba"/>
<DatasetType datasettype_name="Text" elemId="id6" island_id="0"/>
<ImanVolume elemId="id7" island_id="0" volume_name="volume1"/>
<Tool elemId="id8" island_id="0" object_name="TextEditor"/>
<POM_imc elemId="id9" island_id="0" site_id="-2132200621"/>
```

The Importing Teamcenter Site's *site_id*.

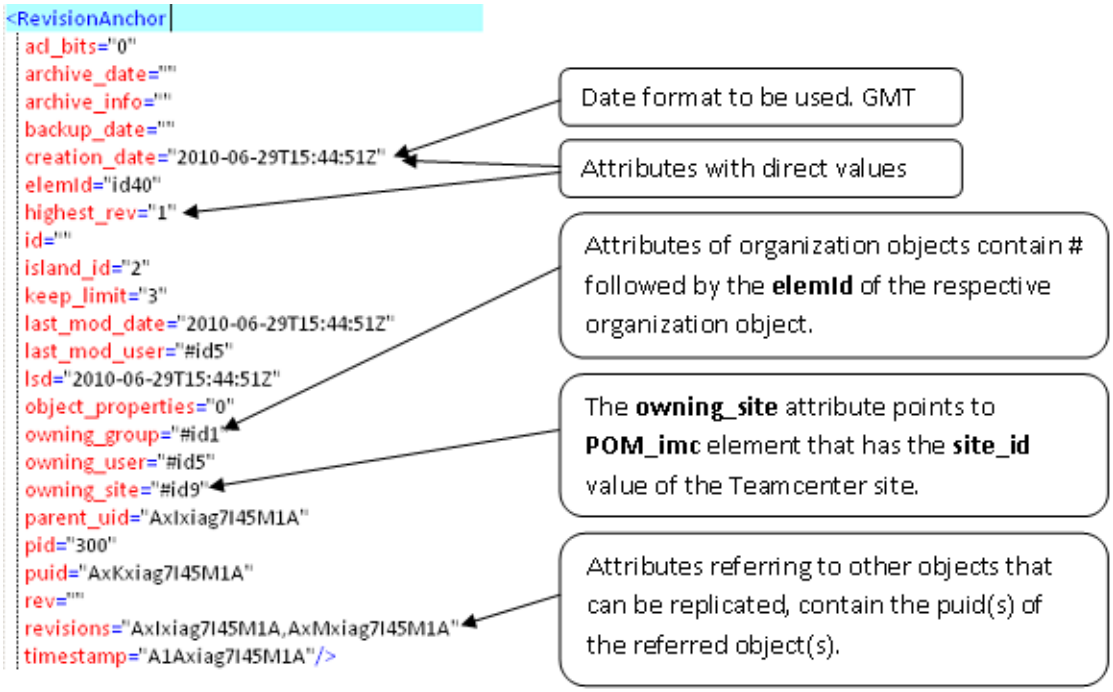
The candidate keys for certain classes are defined by the schema as shown in the following table. The attribute descriptor is **POM_attr_is_candidate_key**. Additionally, the following list of classes and corresponding candidate keys is processed by the fast import function. Some special cases, indicated by a hash (**#**), involve composite keys for the XML element has a dummy attribute containing the composite key values.

Class name	Candidate key
ClosureRule	**name**
RevisionRule	**object_name**
ImanVolume	**volume_name**
Tool	**object_name**
TC_Project	**project_id**
TransferMode	**object_name**
TransferOptionSet	**object_name**
CFM_date_info	**id**
AM_ACL	**ACL_Name**
ImanActionHandler	**handler_id**
ImanEventType	**eventtype_id**
ImanAliasList	**alName**
Condition	**condition_name**
ImanQuery	**query_name**
TaskType	**tasktype_name**
Participant Subclasses*	**assignee**
GroupMember#	**member**
[**Group.name**,**User.user_id**, **Role.role_name**]	
ResourcePool#	**pool**
[**Group.name**, **Role.role_name**, **ResourcePool.allow_sub_groups**, **ResourcePool.all_members**]	
VerificationRule#	**vrule**
[**functionality**, **type**, **context_filter**, **condition_reference**, **subGroup**]	

The XML elements for objects that can be replicate have a list of attributes defined in the schema with the **parent_uid** attribute that indicates the primary object. The list of attributes includes the ones defined directly on the leaf class plus those that are defined on each parent class up to and including the **POM_object** class. In an island, the primary object has an empty **parent_uid** attribute and all the dependent objects have the **parent_uid** value set to the **puid** of the primary object (**parent_uid=puid**).

The objects of one island must appear together in the TC XML file. If they are scattered or otherwise in error, the import of that island fails. (Other islands will be imported.) After the import of an island fails, correct the issues and reimport the failed island.

Following are the XML elements for replicable objects:



Valid values for logical attributes are **Y** or **N**.

If the **owning_site** attribute is missing, the object is created as a replica. The **site-id** value in the **originatingSite** attribute of the **Header** element is considered to be the owner of the object.

For array attributes, the values must be separated by the delimiter defined by the **GMS_tcxml_string_separator** preference. The default delimiter is a comma (**,**). The **Header** element contains details of the originating site, option set or transfer mode used, transfer option name-value pairs, and session option name-value pairs. The **originatingSite** attribute contains the **site-id** value representing the legacy system and must be unique (cannot be the same as the Teamcenter site). This site must be defined in the Organization application of the importing Teamcenter site.

Following is a typical TC XML **Header** element

```
<Header author="infodba" date="2010-06-29" elemId="id44" originatingSite="-2140766078"
        targetSite="" time="21:22:11">

 <TransferFormula elemId="id45">
  <OptionSet elemId="id46" name="SiteConsolidationDefault">
   <Option elemId="id47" name="internalClosureRule" value="True"/>
   <Option elemId="id48" name="opt_entire_bom" value="True"/>
   <Option elemId="id49" name="opt_entire_mse" value="False"/>
   <Option elemId="id50" name="opt_ixr_islanchor" value="False"/>
   <Option elemId="id51" name="opt_mechatro" value="False"/>
   <Option elemId="id52" name="opt_res_audit" value="True"/>
   <Option elemId="id53" name="opt_res_checkout" value="False"/>
   <Option elemId="id54" name="opt_varexp_islanchor" value="True"/>
  </OptionSet>
  <SessionOptions elemId="id55">
   <Option elemId="id56" name="fastStream" value="yes"/>
  </SessionOptions>
  <TransferMode elemId="id57" object_name="SiteConsolidationDefaultTM"/>
  <Reason elemId="id58"></Reason>
 </TransferFormula>

</Header>
```

Using the csv2tcxml utility to create TC XML bulk data for import

Migrate bulk data using CSV files

You can migrate bulk parts data from legacy or other systems to Teamcenter with CSV (Comma Separated Values) files using the following process:

1. Extract the data in CSV format (using SQL or other tools in that system). CSV files must meet the requirements described in **Preparing CSV files**.

2. Convert the CSV data into TCXML using the **csv2tcxml** (CSV-to-TC XML) utility.

3. Import the TCXML file into Teamcenter using **the bulk loader utilities**.

The *csv2tcxml* utility reads one or more CSV files containing parts data and related metadata and generates a TC XML file. The *csv2tcxml* utility is located in the *TC_DATA\csv2tcxml\csv2tcxml* directory and **must be installed before its first use**.

Acquiring a migration ID

Contact your Siemens Digital Industries Software representative to acquire a migration site ID. This ID is required to run the *csv2tcxml* utility.

Supported objects and classes

The *csv2tcxml* utility supports a broad range of objects and classes. See **Supported classes** for a list of supported objects.

Customizing the utility

The utility is highly configurable using the *csv2tcxml.ini* file and supporting *.ini* files. Additional templates are supported by customizing the extendable framework. See the **Customizing the csv2tcxml utility** for more information.

CSV files and Excel

If you open a CSV file in Excel to view its contents, do not save the file as a *.csv* file from Excel. Excel may add extra characters when the file is saved that prevent the **csv2tcxml** utility from converting it successfully.

Install and set up the csv2tcxml utility

Use the following steps to install the **csv2tcxml** utility before using it the first time.

1. Start a Teamcenter command shell as a Teamcenter administrator and change to the *TC_DATA\csv2tcxml* directory.

2. Generate a data model from your templates by entering a command of the following form.

   ```
   csv2tcxml install Tc-admin-user password group template_folder -t
   template_name -c
   ```

   where:

Tc-admin-user password group	Optional. Siemens Digital Industries Software recommends providing Teamcenter administrator credentials when working with **ImanFile** objects. If credentials are given, *Tc-admin-user*, *password*, and *group* must each be given.
template_folder	Optional. Specifies the full path to a directory containing templates. By default, the *TC_DATA_MODEL* directory is used.
-t *template_name*	Optional. Specifies a template from which the utility generates the data model. This argument can be used multiple times. If not specified, the data model is based on all templates in the *TC_DATA_MODEL* directory. Each template's *template_name_dependency.xml* file must exist in the same directory as the template file.
-c	Optional. Generates an HTML report defining the classification hierarchy. This information can be very useful when working with classification objects.

The **-c** argument requires you run the **csv2tcxml** utility with administrator credentials.

> Note:
>
> If you are not working with a Teamcenter environment, but have a folder containing templates, use a command of the following form to generate a data model from the templates.
>
> ```
> csv2tcxml install Templates_Folder [-t template_name]
> ```
>
> Where **Templates_Folder** is the folder containing the template files.

3. In a text editor, open the file *TC_DATA\csv2tcxml.ini*. Set **source_site** to **the migration ID provided by Siemens Digital Industries Software**.

Files created by the installation

Running the **csv2tcxml** utility as shown with the **install** argument generates the following data model files in the *TC_DATA\csv2tcxml\model* directory:

csv2tcxmltc_datamodel.xml

Data model overrides and definitions used by the **csv2tcxml** utility.

classification.html

(Optional.) A formatted summary of the classification object class hierarchy of the new data model. This report can be helpful when extracting classification data to create CSV files. Be aware that is a class has no key definition, in may not be included in the summary.

datamodel.html

A formatted summary of the class hierarchy of the new data model. This report can be helpful when extracting data to create CSV files.

tc_admin_data.json

This file is created only when Teamcenter administrator credentials were provided when running the **csv2tcxml** utility. *tc_admin_data.json* contains the Teamcenter administration data used by the **csv2tcxml** utility to set **ImanFile** object attributes (such as **volume_tag** and **sd_path_name**) automatically.

tc_datamodel.xml

The data model information created based on the specified template files.

Converting CSV files to TC XML

Preparing CSV files

CSV column names are required to be in the format *ClassName*:*AttributeName*. See any of the CSV files in the *TC_DATA\csv2tcxml\examples* directory for examples. The example files use the pipe (**|**) character as the value separator.

Review the data model summary report *TC_DATA\csv2tcxml\model\datamodel.html* for class and attribute relationships. Review the data model summary report *TC_DATA\csv2tcxml\model\classification.html* for class and attribute relationships.

Ensure your CSV files meet the following requirements.

CSV standard

The CSV header is required when using the *csv2tcxml* utility.

Aside from requiring the header, imported CSV files must adhere to **the RFC 4180 technical standard** to work with the *csv2tcxml* utility. (RFC 4180 defines the CSV header as optional.)

Use a comma as a separator in the header. A CSV file with a header beginning with an exclamation point (formerly required by the utility) generate a warning.

You may be able to simplify CSV creation by setting the **sep** and **quote** parameters to custom values. For example, in your data, you may have a string with a comma and a double quote such as **'12" CYL ASSEMBLY, APPROVED'**. According to the CSV standard, you must represent this string as **"12"" CYL ASSEMBLY, APPROVED"**. However, if you set the following two parameters as follows, you do not need to change the string.

```
sep=#
quote=$
```

Setting these values specifies that **#** is the separator and **$** is the quote character. As a result, the conflict is avoided and no quote or escape is needed as long as you use **#** as the separator throughout the CSV.

Experiment with different separator and quote characters to avoid quoting and escaping. This approach is much easier than reformatting the data.

See **Customizing the csv2tcxml utility** for details on these and other parameters.

File encodings

By default, the *csv2tcxml* utility processes CSV files as UTF-8 encoded files. The *csv2tcxml* utility supports several CSV file encoding standards in addition to UTF-8. To use files with other encodings, specify

the encoding standard using the encoding value in the *csv2tcxml.ini* file. See **Supported encoding standards** for a complete list of supported standards.

TC XML output files are always encoded as UTF-8.

Commas and double quotes in data

If your data strings contain commas, enclose the data strings in double quotes. For example: **aaa,bbb,"cc,c"**

If your data strings contain double quotes, quote the data strings first, and then escape the double quote by preceding it with another double quote. For example: **aaa,"b""bb",ccc**

To simplify data formatting, you can configure the *csv2tcxml* utility to process CSV data that uses characters other than commas and double quotes to format data. For example, when extracting your data to CSV format, you could choose to use the **#** character as a field separator and the **$** character as the quote character.

To properly convert CSV files with these formatting characters, in the *csv2tcxml.ini* file set the **sep** and **quote** parameters to the characters acting as separators and quotes in the data. For example: **sep=#** and **quote=$**.

Using alternative characters in the CSV file can be helpful when your data contains strings such as **'12" CYL ASSEMBLY, APPROVED'**. The CSV standard requires this string be formatted as **"12"" CYL ASSEMBLY, APPROVED"**.

Using the alternative characters set in the example means the example data string would not need to be changed to conform to the CSV standard. When using **#** as the separator and **$** as the quote character in the CSV file, there is no conflict with the commas and double quotes in the string.

Creating CSV files using alternative characters to format the data can be significantly more efficient than reformatting data to match the default CSV comma and quote requirements. The CSV files in the *TC_DATA\csv2tcxml\examples* directory use the pipe (**|**) character as the value separator.

For further information on setting parameters, refer to **Customizing the csv2tcxml utility**.

Convert CSV files to TC XML

Use the following steps to convert a CSV file containing parts data to TC XML.

1. Start a Teamcenter command shell as a Teamcenter administrator and optionally change to the directory containing the CSV file.

2. Enter a command with the following form to convert the CSV file to TC XML.

```
csv2tcxml csv-file-name -p param-name=param-value -o output-dir
```

where

csv-file-name

The name of the CSV file containing the parts data to convert to TC XML. Include the full path to the file if it is not in the current directory.

-p

Optional. Overrides the *csv2tcxml.ini* parameter *param-name* with the value *param-value*. This argument can be used multiple times.

For further information on parameters, refer to **Customizing the csv2tcxml utility**.

-o

Optional. Specifies a directory *output-dir* in which to save the resulting TC XML file. By default, the file is saved in the current directory.

Results of the conversion, along with any error messages are displayed and saved in a log file in the current directory.

3. Import the TC XML data into Teamcenter using the **tcxml_import** command with the **-bulk_load** argument.

Converting objects

Object name - class name with tag

In the CSV header, *classname*:*attribute-name* is typically given as a column name. However, one row can create multiple objects of the same class. Master forms are and example of this case. One master form exists for **Item** and one master form for **ItemRevision**. Although they have different types, they typically are all of **Form** class. Use object name to override an attribute on one of the forms.

To use object name, set each column as *object-name*:*attribute-name*. Object name is composed of the class name and an optional tag of the form **Dataset[***UG***]**, where *UG* inside the brackets is a tag. The tag can be any word characters (**[-__0-9a-zA-Z]+**).

With this approach, you can create multiple objects of the same class in a single row. Refer to the example *TC_DATA\csv2tcxml\examples\foundation\Item_two_DS.csv*. In that file, two **Datasets** are created for each row using a header of the following form:

```
Item:item_id,Item:object_type,ItemRevision:item_revision_id,Dataset[UG]:d
ataset_type,Dataset[UG]:object_name,Dataset[JT]:dataset_type,Dataset[JT]:
object_name
```

Add a name tag only when necessary, as class name alone can also be the object name.

Object type

The **csv2tcxml** utility typically creates an object based on the type, not the class. From the type, the class can be determined data model. If only the class name is given in the header, the utility can still determine the type.

1. If there is a type with **typeClassName** and **typeName** the same as the header class name, that type is used.

2. Otherwise, the utility will find all types with **typeClassName** is equal to the header class name. If only one type is found, that type is used. Otherwise, an error is reported.

You can provide type information through an extra column using the form **Item:object_type**. This is referred to as a dynamic type. For example:

```
Item:item_id,ItemRevision:item_revision_id,Item:object_type
CSV_00001,A,Design
CSV_00002,A,Part
```

Doing so creates two different types of **Item** in one file. The item for the first row is **Design**. The item for the second row is **Part**.

If the **Item** type is fixed in the file, author the CSV as follows:

```
Design:item_id,Design_0_Revision_alt:item_revision_id
CSV_00001,A
CSV_00002,A
```

For this case, the **Item** will always be **Design**.

Value validation

An attribute can be one of the following types:

- String
- Integer
- Double
- Date
- Logical
- Reference

The attribute type is defined in the data model. The **csv2tcxml** utility also handles validation of VLA (variable length array) and LOV (list of value) values.

- If an attribute is a VLA or fix array, each value is checked without exception.
- If a fixed array attribute is missing values, empty values are appended. If too many values are given, an error is reported.
- Be aware that an empty string is always a valid value.

String

For string attributes, a max string length check is performed. If a max string length is defined in the data model, the utility checks if the value exceeds that limitation. If a given value exceeds that length, the value is truncated and a warning is logged. This check and truncation cannot be bypassed.

If an LOV is attached to an attribute, the value must be a valid value in the LOV. If it is not a valid value, the value is set and an error is logged.

The max string length is measured using database encoding in byte length, not in character length. Therefore, one character may occupy a different number of bytes in different databases with different encodings. The utility assumes utf8 encoding for database encoding unless the **db_encoding parameter** is set otherwise.

Integer

If the value is not an integer, an error is logged and the value is reset to be empty.

If an LOV is attached to an attribute, the value must be a valid value in the LOV. If it is not a valid value, the value is set and an error is logged.

Double

A double value can be a value such as **-123.4567e-89** or an integer such as **123**. If it is not a valid value, the value is reset to empty and an error is logged.

Date

By default, the following date format is supported:

```
YYYY\Dmm\Ddd[\DHH\DMM\DSS]?
```

\D means any non-numeric character. Time (**\DHH\DMM\DSS**) is optional. If a time is not given, **00:00:00** is set. Following are example dates:

```
2018-12-06 9:10:8
2018/12/6-9:10:8
2018-12-6
```

The utility converts them to 2018-12-06T09:12:13Z and 2016-12-06T00:00:00Z respectively (both are GMT times).

To use an alternate date format or to perform a time zone conversion, set the **local_time_zone** parameter a value. A value of **-500** represents the UTC-05:00 Eastern time zone. A value of **+900** represents the UTC+09:00 time zone. **GMT** represents the Greenwich Mean Time zone. Setting the **local_time_zone** parameter allows the utility to recognize the following additional time formats:

Format	Description
"Wed, 09 Feb 1994 22:23:32 GMT"	HTTP format
"Thu Feb 3 17:03:55 GMT 1994"	ctime(3) format
"Thu Feb 3 00:00:00 1994"	ANSI C asctime() format
"Tuesday, 08-Feb-94 14:15:29 GMT"	old rfc850 HTTP format
"Tuesday, 08-Feb-1994 14:15:29 GMT"	broken rfc850 HTTP format
"03/Feb/1994:17:03:55 -0700"	common logfile format
"09 Feb 1994 22:23:32 GMT"	HTTP format (no weekday)
"08-Feb-94 14:15:29 GMT"	rfc850 format (no weekday)
"08-Feb-1994 14:15:29 GMT"	broken rfc850 format (no weekday)
"1994-02-03 14:15:29 -0100"	ISO 8601 format
"1994-02-03 14:15:29"	zone is optional
"1994-02-03"	only date
"1994-02-03T14:15:29"	Use T as separator
"19940203T141529Z"	ISO 8601 compact format
"19940203"	only date
"08-Feb-94"	old rfc850 HTTP format (no weekday, no time)
"08-Feb-1994"	broken rfc850 HTTP format (no weekday, no time)
"09 Feb 1994"	proposed new HTTP format (no weekday, no time)
"03/Feb/1994"	common logfile format (no time, no offset)
"Feb 3 1994"	Unix 'ls -l' format
"Feb 3 17:03"	Unix 'ls -l' format
"11-15-96 03:52PM"	Windows 'dir' format

If the date string is not a valid value, the value is reset to empty and an error is logged.

Note:

Local-to-GMT time conversion is supported for dates between 1901-12-17 00:00:00 GMT and 2038-01-16 23:59:59 GMT. If a date outside of that range and **local_time_zone** is set, provide the date in GMT to avoid the local-to-GMT conversion. For example, provide a date value such as **9999-12-30Z**. The suffix **Z** specifies the date is in GMT. The resulting date string is **9999-12-30T00:00:00Z**.

Logical

Valid logical values are **true**, **false**, **y**[**es**], **n**[**o**]. Values are case-insensitive. For example, **True**, **False**, **Y**, and **n** are all valid values. The value, '**not**' is invalid.

If a value is not valid, the value is reset to empty and an error is logged.

Reference

Typically, a reference attribute is provided by a PUID directly or with an object's GSID string. See **Object GSID and PUID values** for details on those values.

You can also provide an ID for an object you want to reference in the same file and then reference that object using its ID. See **Object ID** for more information on working with object IDs.

To refer to an admin object, you must also provide the candidate key value. See **Admin object and candidate key** for more details.

LOV

If an attribute has a list of values attached, those values are the attribute's valid values. The utility checks the attribute value against the values in the attached list. If a value is not in the list, the value is remains set as it is and an error is logged.

Note:

For cascading LOVs, the relationship between parent property value and child property value is not verified. For a level 2 property such as this, all of its possible values are valid without checking its parent's property value.

Use the **lov_validate** parameter to skip LOV checking. Setting **lov_validate** to **0** will bypass the LOV validation. See **Customizing the csv2tcxml utility** for more details.

Admin object and candidate key

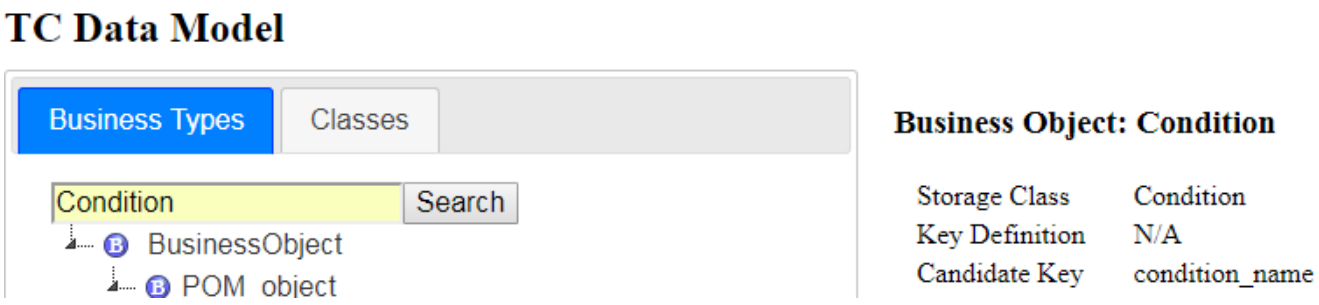
TC XML has two types of data – administration (admin) data and transfer data. Admin data is not transferred. The TC XML import always searches the admin objects in the target system. Admin data is in island 0 in the XML before transfer data begins.

Every admin object class has a candidate key defined. For example, **user_id** is the key for **User** and **project_id** is the key for **TC_Project**. Refer to **Admin classes and candidate attributes** for a full list of admin objects and their candidate attributes.

You can extend or define your own admin class if needed. See **Overview of extending the data model**.

You do not need to create an admin object in the CSV file. When an admin object is needed, you can provide its candidate key value.

Before you provide a value, ensure you know it is an admin reference attribute and its candidate key. Find the candidate key for a type or class by opening *TC_DATA\csv2tcxml\model\datamodel\datamodel.html*. Search and select the type or class. The candidate key is shown if one exists:



For example:

```
Item:item_id,ItemRevision:item_revision_id,Item:owning_user,ItemRevision:
owning_user,ItemRevision:owning_group
Item00001,A,tcadmin,chenji,Engineering
Item00002,A,chenji,tcadmin,dba
```

In this example, the **csv2tcxml** processes the reference based on the user id and group id in the corresponding columns and checks the data model to see if a reference attribute is related to an admin type.

> Note:
>
> One special case exists where an attribute can reference different types of admin objects. For example, **ScheduleMember.resource_tag** is a typed reference pointing to **POM_object**. When used, it can refer to a Group or to a User. To specify which is being referred to, specify the value as in **dba@Group** and **migration@User**. The string before the **@** character is the candidate key value. The string after the **@** character is the admin class name.

Principal object and its helper

Principal object are objects that have MFK (multifield key) constants defined or key attributes defined. Before converting the data, the **csv2tcxml** utility checks the CSV header to identify the principal objects.

For example:

```
Item[A]:item_id,ItemRevision[A]:item_revision_id,Item[B],ItemRevision[B]:
item_revision_id
```

In this header, **Item[A]** and **Item[B]** are principal objects, because **Item** has MFK defined with **Item{item_id}**. **ItemRevision[A]** and **ItemRevision[B]** are not principal objects. The utility only knows if the two columns (**ItemRevision[A]** and **ItemRevision[B]**) are used when processing each row.

For each row, the utility creates the principal objects one by one. After object creation, the following steps occur:

1. Go through all the columns to see if a value applies. If either of the following conditions matches, the value is assigned to the object:

   - The current object has the same name as the column's **object name**.
   - The column's object name is a class name and this object is of the same class or its subclass.

2. Create helper objects defined for this class.
3. Call object extension **onCreate**.
4. Apply initial values retrieved from the data model for the current object.
5. (Additional steps also occur.)

In the earlier example when **Item** is created, because **Item** has the **onCreate** extension registered, an extension function is called. With that extension, **ItemRevision** is created and that new object goes through the same steps as **Item**. The extension then links **Item** and **ItemRevision** to each other.

A helper object may have its own helper object defined, for example master form for **ItemRevision**, storage class for forms, and so on. Principal object creation may trigger several objects to be created in succession. After principal objects are created, the object creation for the row is done.

Consider the following link in a CSV file:

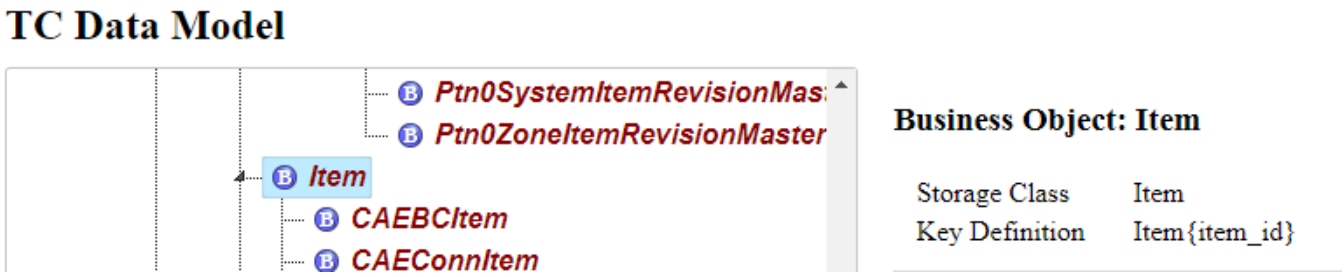
```
Mdl0DefaultGeometry:mdl0ext_transform_rot00,Mdl0DefaultGeometry:mdl0ext_t
ransform_rot10
```

No object is created because there is no principal object. This approach is similar to creating an object in Teamcenter, where you cannot create Default Geometry alone; you must create its principal object – Design Element. That does not mean **Mdl0DefaultGeometry** cannot be created.

Besides MFK objects, the *csv2tcxml_data_model.xml* file also defines **keyAttributes** for certain classes to create principal objects. For example:

```
<TcClass className="Mdl0ApplicationModel" keyAttributes="mdl0model_id"/>
```

To check if a type or class is a principal object, you can open the *TC_DATA\csv2tcxml\model\datamodel.html* page generated during installation. Search for the type or class in the left tree panel to see its key definition. For example:



Whenever an object is created, the **csv2tcxml** utility checks if a helper object is defined for the object. In *csv2tcxml_datamodel.xml*, the following is defined:

```
<TcClass className="Mdl0ConditionalElement"
helper="Mdl0ElementThread(&lt;-$.mdl0element_thread;&lt;-$.wso_thread)"/>
```

For each **Mdl0ConditionalElement** (including subclasses) object, the utility creates an **Mdl0ElementThread** and links it to the main object (**Mdl0ConditionalElement**). Refer to **Object linking**.

Create helper objects using one of the following methods:

- Define the helper for a class in *csv2tcxml_data_model.xml*.
- Define a linking rule for an object in the header. Refer to **Object linking**.

  If you want to create an extra **Form** object and attach it to an **ItemRevision**, consider the following line:

  ```
  Item:item_id,ItemRevision:item_revision_id,Form[CostData]:object_type(<
  -IMAN_Reference<-ItemRevision)
  CSV_00002,A,CostDataForm
  ```

  The last column contains a linking rule in header. This instructs the utility to create that object and link it to some other object. If you remove the linking rule from the last column, the extra **Form** is not created, as **Form** is not a principal object.

- Create an extra helper by creating **onCreate** or **postCreate** extensions. (However, Siemens Digital Industries Software does not recommend using this approach.)

Object linking

The **csv2tcxml** utility does not only create the objects, it also creates links to objects. Linking rules specify how the objects are connected. Generally, objects are linked through reference attributes or **ImanRelation** GRM (generic relationship management) objects.

Using GRM, the relationship is a one-to-one link. Two formats exist for writing rules linking two objects by GRM:

- **PrimaryObjectName**->**RelationType**->**SecondaryObjectName**
- **SecondaryObjectName**<-**RelationType**<-**PrimaryObjectName**

Reference attributes have one-to-one or one-to-N relationships. Two formats represent this kind of link:

- **MainObjectName.attribute_name**->**ReferencedObjectName**
- **ReferencedObjectName**<-**MainObjectName.attribute_name**

You must create a portion of these linking rules. Create links using the following methods:

- Provide a linking rule in the header.
- Provide a linking rule in a class's helper.
- Provide a linking rule in an initial value.
- Link using a GSID or PUID.

Provide a linking rule in the header

In the CSV header, append linking to an object:

Header column	Linking rule generated	Comment
P:a(->R->S)	P->R->S	GRM forward linking
S:a(<-R<-P)	S<-R<-P	GRM backward linking
P:a(b->S)	P.b->S	Attribute forward linking
S:a(<-P.b)	S<-P.b	Attribute backward linking

The **csv2tcxml** utility uses the object name as the left-most part of the linking rule. You must write the remaining part of the rule in parentheses. Multiple rules may exist for one name as shown in the following example:

```
Pdm0ExternalModel:pdm0ModelId(pdm0TargetModel->Cpd0CollaborativeDesign;<-Pdm0RelatedModel<-Prg0ProgramPlan),Cpd0CollaborativeDesign:mdl0model_id,Prg0ProgramPlan:prg0PlanId;Prg0ProgramPlan:object_name
0x0001,CD00001,PRG0001
```

Separate multiple rules with semicolons. The example header generates two rules:

```
Pdm0ExternalModel.pdm0TargetModel->Cpd0CollaborativeDesign
Pdm0ExternalModel<-Pdm0RelatedModel<-Prg0ProgramPlan
```

This type of linking rule is executed after all objects are created for a row.

Provide a linking rule in a class's helper

In the CSV data model file, you can define helpers for a class. When you write a concrete class as a helper object, you typically link that helper object to the main object. In the helper rule, **$** represents the main object which triggers this helper creation. For example:

```
<TcClass className="Mdl0ConditionalElement" helper="Mdl0ElementThread(&lt;-$.mdl0element_thread;&lt;-$.wso_thread)"/>
```

(This rule is defined in XML, so you must write **&lt** for **<**.)

The linking rule is enclosed in parentheses and delimited by semicolons. Two linking rules are generated for this example:

```
Mdl0ElementThread<-Mdl0ConditionalElement.mdl0element_thread
Mdl0ElementThread<-Mdl0ConditionalElement.wso_thread
```

These linking rules are executed immediately after the helper's creation.

One main object may have multiple helpers that can be linked together.

```
A(a->$),B(<-$.b;->R->A)
```

This rule generates two helper objects A and B. The following link rules are also formed:

```
A.a->$
B<-$.b
B->R->A
```

Both A and B are linked to the main object and a GRM links A and B.

Multiple helper objects are separated by commas.

Provide a linking rule in an initial value

The linking rule may also be used in the initial value for a specific attribute as in the following example.

```
<TcClass className="Mdl0ModelElement">
    <TcAttribute attributeName="mdl0model_object" initialValue="(-
>Mdl0ApplicationModel)"/>
</TcClass>
```

The rule generated is **Mdl0ModelElement.mdl0model_object->Mdl0ApplicationModel**. Note that the rule is enclosed in parentheses.

Since the left part (**Mdl0ModelElement.mdl0model_object**) is already known, you only need to write the remaining part. Only one kind of link rule is supported here – forward attribute linking.

This type of rule is executed with the header linking rules after object creation.

Link using a GSID or PUID

Linking rules are not the only way to link objects. You can also directly provide a value to a **reference attribute**. Unlike an admin object reference for which you provide a candidate key value, you can provide a **GSID or PUID** or an **Object ID**.

Object GSID and PUID values

The **csv2tcxml** utility creates a unique PUID for each object based on a generated GSID and your source site ID. The same PUID is created each time the same source site ID and same GSID are used. PUIDs must be unique in a Teamcenter database. Objects cannot be imported if they have IDs identical to those of existing objects.

With typical migration projects, this PUID generation approach is adequate. However, when performing migrations of data to multiple sites, contact your Siemens Digital Industries Software representative to acquire a new source ID for the additional sites. Doing so will ensure that each site has objects with different PUIDs, allowing for later data exchange

The GSID string is composed of certain key attribute values and the object type name. The object type is always be part of the GSID string.

Key attributes for an object can be the multifield key (MFK) values or keyAttributes defined for the class. For example, if Item has an MFK defined as **Item{item_id}**, or Item 000100, its GSID string is **000100!Item**. For Design 000100, the GSID is **000100!Design**.

For helper objects, the GSID is a combination of the main object's GSID, certain attribute values, and the object type name. The main object's GSID is included to avoid possible PUID clashes.

You can find every object's GSID by setting **the save_gsid_out parameter**. With **save_gsid_out** set to **1**, a *gsid.out* is generated during conversion. This file contains the PUID/GSID pairs for all objects.

You can also query the PUID for an existing object in the database as described in **Query the Teamcenter database for existing objects**. If the object is found, the **csv2tcxml** utility does not perform a GSID hash for the object and the object's PUID is used directly.

Avoiding GSID clashes

For objects having MFKs defined, the GSID should never clash. Other objects, however, may have duplicate GSID values. For example, you may create two objects of the same type with each attribute (except the PUID) having the same value. In this scenario, the **csv2tcxml** utility creates the same GSID for each object, creating a GSID clash.

To work around this rare occurrence, use the custom attribute **gsid_suffix** when creating the *.csv* file.

Set the column name as *object-name*:**gsid_suffix**. If *object-name* is a class name, its value does not apply to any subclasses. The composition of the object GSID follow the standard rules. If an object has **gsid_suffix** supplied, the value specified for **gsid_suffix** is appended to the GSID string.

> Caution:
>
> Use the **gsid_suffix** attribute only when there is no other way to distinguish two objects through their attribute values. **gsid_suffix** is valid for any object. Do not use **gsid_suffix** for objects with MFKs defined.
>
> It is your responsibility to ensure the composed GSID is actually unique when created using the **gsid_suffix** value.

For example, the following input creates two datasets of the same type attached to the same **ItemRevision:**

```
Item:item_id,ItemRevision:item_revision_id,Dataset:dataset_type,Dataset:o
bject_name(<-IMAN_reference<-ItemRevision),Dataset:gsid_suffix
CSV_00001,A,Text,CSV_00001/A,Unique_id1
CSV_00001,A,Text,CSV_00001/A,Unique_id2
```

Since the key attribute for Dataset is "**$parent,object_name**", and those two datasets have the same parent object (**ItemRevision**) and **object_name** (**CSV_00001/A**), then those two objects have the same GSID unless a **gsid_suffix** value is also provided.

> Example:
>
> Consider two instances for a class that have the same values for all their attributes except for their PUID values. You can set **gsid_suffix** as its key attribute to make that class a principal class. The final GSID string for the object would be "*!TypeName***!***gsid_suffix_value*".

Object ID

The **csv2tcxml** utility processes data line-by-line. The utility does not retain information about completed lines. However, you have the following options if you need to reference an object generated in previous line.

- If the referenced object is a principal object, you can reference that object using key values. This approach requires changes to the file header and can be complex.
- If the referenced object is a helper object, for example a **PSOccurrence** object generated in a BOM CSV file, you can reference it using the custom attribute **_id_**.

  Set the **_id_** attribute like any other attribute for an object. The value can be any string that is unique within the conversion.

  During conversion, the **csv2tcxml** utility binds the object ID to its PUID after processing the line. Then later lines can reference that object using **{ID}**. For example,

```
Item:item_id,ItemRevision:item_revision_id,ItemRevision:_id_,ImanRelati
on:relation_type(pimary_object->ItemRevision),ImanRelation:secondary_ob
ject
000023,A,000023/A
000023,B,000023/B,IMAN_based_on,{000023/A}
```

  The first line creates **ItemRevision** A for **000023** and binds an ID (**000023/A**) to it. In a later line, revision B is created and linked with revision A by a GRM.

  With the object ID, a cross line reference can be created.

Cross file reference and multiple input files

Object ID is not limited to one file. The **csv2tcxml** utility supports multiple input files allowing you to set up references across files. This approach supports complex data models such as those used for service planning and bill of process which require multiple windows. Describing all windows in a single files is extremely difficult. Splitting the data into multiple CSV files and cross referencing files is a more realistic approach.

When referencing across files, all of the input files must be converted using a single command line using the following form:

```
csv2tcxml input1.csv input2.csv [-o output_file.xml]
```

As an example, the first CSV file creates the **Item** and **ItemRevision** objects.

```
Item:item_id,ItemRevision:item_revision_id,ItemRevision:_id_
000023,A,000023/A
000023,B,000023/B
```

The second file links to the **ItemRevisions**:

```
ImanRelation:relation_type,ImanRelation:primary_object,ImanRelation:secon
dary_object
IMAN_based_on,{000023/A},{000023/B}
...
```

The CSV files are processed in the order they are given in the command, so files must be listed in the proper sequence. That is, the CSV file containing the relationships cannot be listed before the CSV creating the objects in this example.

If the **-o** option is not given, a single output file will be generated. Otherwise each input CSV file will have its own output XML file in the directory specified by **-o**.

Aggregation class

If a class is defined as an aggregation class, you can create multiple objects by providing multiple values for one column. Three classes are defined as aggregation classes: **ImanFile**, **Fnd0TableRow**, and **ReleaseStatus**.

Consider **ImanFile** as an example:

```
Dataset:dataset_type,Dataset:object_name,ImanFile:original_file_name
UGMaster,000111,"000111.prt,000111_images_preview.qaf"
UGMaster,000112,000112.prt
```

ImanFile.original_file_name is a single value attribute. However, you can give multiple values to create multiple objects. For example, one **ImanFile** has "**000111.prt**" in **original_file_name**. The other has "**000111_images_preview.qaf**" as a value.

The number of objects created by the utility depends on the supplied value. For example,

```
ImanFile:original_file_name,ImanFile:volume_tag
"000111.prt,000111_images_preview.qaf",volume
```

In this example, **original_file_name** has two values and **sd_path_name** has only one value. Two **ImanFile** objects will be created, both having the same **volume_tag** of "**volume**".

Attribute propagation

After an attribute is set, you may need to assign the same value to other objects. For example, when you set **Item:owning_user**, you may also want its **ItemRevision** and master forms to have the same **owning_user** value. This can be achieved by attribute propagation.

Propagation rule

Propagation rules are defined In the data model. The rules are organized into groups such as **Security Group I**, **Security Group II**, and so on. These rule groups are attached to properties using **PropertyConstantAttach**.

This **csv2tcxml** utility analyzes the propagation information during conversion. If an attribute is set and it has a propagation rule group attached, the utility runs the corresponding propagation rules to propagate the value to traversed objects and creates a **ProjectObjectRelation**.

Five properties have propagation rule groups attached: **ip_classification**, **gov_classfication**, **license_list**, **project_list**, and **owning_project**. Each of these is defined on **WorkspaceObject**.

For example:

```
Item:item_id,ItemRevision:item_revision_id,Item:project_list
001,A,Athena
```

Item.project_list is set to "**Athena**". Since **project_list** has a propagation rule attached, the utility propagates **project_list** from **Item** to **ItemRevision**. Both **Item** and **ItemRevision** then have the project assigned.

Following is a more complex example:

```
Item:item_id,ItemRevision:item_revision_id,Item:project_list,ItemRevision
:project_list,Dataset:object_name(<-IMAN_specification<-ItemRevision),Dat
aset:dataset_type
001,A,PRJ1,"PRJ2,PRJ3",001-UGMaster,UGMASTER
```

In this example, **Item** is assigned to project PRJ1 and **ItemRevision** is assigned to projects PRJ2 and PRJ3. After both attributes are set, the utility runs the propagation resulting in PRJ1 being propagated to **ItemRevision** and **Dataset** and PRJ2 and PRJ3 being propagated to **Dataset**. **Item** is assigned to PRJ1. **ItemRevision** and **Dataset** are assigned to PRJ1, PRJ2, and PRJ3.

In the latter example, the final value of **project_list** is merged. The utility process rules according to their types:

Type	Result
Merge	Combine the values (VLAs only).
Fill	If the propagated object does not have a value for the attribute, fill it. Otherwise, set no value.
Override	Always set the value.
Ordered	If the propagated value has an equal or higher order, set the value. Otherwise, propagate without setting the value. **PropagationRule** defines the value order.

Customized attribute propagation

You may wish to propagate attributes differently than the predefined propagation rules in the data model.

For example, you may want to propagate **owning_user** from **Item** to **ItemRevision**, Master Forms, and datasets attached to **ItemRevision**.

In **csv2tcxml_data_model**, certain classes has **propagateAttributes** defined:

```
<TcClass className="Item" propagateAttributes="owning_user,owning_group"
… />
```

The value is the set of attributes you want to propagate from this object. The targets are its child objects.

As described in **Principal object and its helper**, when a principal object is created, its helper objects are also created. A helper object may have its own helper object. An object tree is created with the principal object as the root.

Ass the relationship is a tree, the utility is aware of the parent and the child. Customized attribute propagation is based on this parent-child relationship.

Object islands

Objects in TC XML are grouped in **islands**. During a data import, the **tcxml_import** utility processes one island at a time. If an error happens for an island, the entire whole island is not imported.

Control the granularity of islands being process with the **csv2tcxml** utility using **the island_batch_size parameter**.

The smallest island is the number of objects specified in a CSV row; you cannot split the objects for one row into multiple islands. To keep each line as an island, set **island_batch_size** to **-1**. Migration performance is improved by grouping more objects (for example, 1000 or 2000) in an island. The default value of **island_batch_size** is **6000**. Be aware that larger islands may make diagnosing import issues more difficult. Consider setting **island_batch_size** is **-1** during your migration development.

Be aware that the number of objects in an island may not be the exact number you set. The object count will be equal to or greater than the value you set, as objects generated for one line will be always in the same island.

Object grouping

By default, all objects for one row are grouped together. You may have a requirement to group different lines together due to your application logic.

Consider the following example:

```
Item:item_id,ItemRevision:item_revision_id
000110,A
...
000110,B
```

If revision A for 000110 is generated at, for example, line 1, and revision B for the same Item is generated at line 500, there is no guarantee that those two ItemRevisions are in the same island. To group these two ItemRevisions together, no matter where they are generated, use **the grouping parameter**. If you were to set **grouping=Item** for this example, objects for line 1 and line 500 are grouped together because they generate the same **Item**.

Note that even though the same items are generated for these two, only one **Item** is written out along with revision A.

The **grouping** parameter has an optional relationship value. This value supports a BOM structure CSV file where each line has parent and child items. Specify **Item[Parent]** or **Item[Child]** to identify the **Item** to be grouped. The parameter supports only one value.

Combined column

Multiple attributes can be given the same value. Instead of copying the value multiple times, the headers can be combined.

```
Item:item_id,ItemRevision:item_revision_id,Item:object_desc;ItemRevision:
object_desc
CSV_00003,A,Test
```

The header appears to have four columns, but it actually has only 3. **Item:object_desc;ItemRevision:object_desc** combines two attributes in one column, providing one value for both attributes.

Use semicolons to combine multiple attributes.

Column alias

Use the **ColumnAlias** section of the *csv2tcxml.ini* file to perform name replacements using aliases:

```
lsd=POM_object:lsd
owning_site=POM_object:owning_site
creation_date=POM_application_object:creation_date
last_mod_date=POM_application_object:last_mod_date
```

The primary purpose for this section is to support naming used in earlier versions of the **csv2tcxml** utility. Siemens Digital Industries Software does not recommend defining your own aliases, but if doing so becomes necessary, use names that are self explanatory and clear.

Query the Teamcenter database for existing objects

If an object not created by the **csv2tcxml** utility already exists in the Teamcenter database, it cannot be directly referenced as its PUID is not derived from the GSID. (See **Object GSID and PUID values**.) However, the **csv2tcxml** utility can query the database to see if an object already exists. If an object is found, its PUID is used in the converted TC XML file. This functionality is also available when using the utility in **update mode**.

To query the database, set the following parameter values in the **QueryDB** section of **the** *csv2tcxml.ini* **file**.

Parameter	Setting
query_db	The class names to be queried.
tc_admin_user	The user ID of a user with Teamcenter administration privileges.
tc_admin_password	The administrative user's password.

When query_db is set as described if an object is found in the database, its PUID is used. If an object cannot be found, the utility generates the PUID from GSID in the standard manner.

Be aware of the following limitations:

- Only principal objects are supported (classes with MFK or **keyAttributes** defined).
- Specified classes must be subclasses of **WorkspaceObject**.
- Key attributes cannot contain reference attributes.

Supported templates and objects

Item and ItemRevision

The **csv2tcxml** utility **supports a broad range of objects and templates**. To see the specific templates and objects supported, review the contents of the *TC_DATA\csv2tcxml\examples* directory. This documentation discusses certain migration scenarios. The example files in that directory cover additional scenarios. Contact your Siemens Digital Industries Software representative with questions about supporting other objects or classes.

Item and ItemRevision

item_id and **item_revision_id** columns support **Item** and **ItemRevision**.

For example:

```
Item:item_id,ItemRevision:item_revision_id
CSV_00001,A
```

Dynamic type example:

```
Item:item_id,ItemRevision:item_revision_id,Item:object_type
CSV_00001,A,Design
CSV_00002,A,Part
```

During converting, **Item** and **ItemRevision** objects are created. For Design, the **ItemRevision** class is **Design_0_Revision_alt**. The form types are "**Design Master**" and "**Design Revision Master**". Information for creating the objects is derived from the data model.

Custom forms

Form has no key attribute defined. It is treated as helper object. To create a custom form, you must attach it to another object, such as **ItemRevision**. The attached object is its parent. For example:

```
Item:item_id,ItemRevision:item_revision_id,Form[Mfg0ArcOverrideForm]:obje
ct_type,Form[Mfg0ArcOverrideForm]:object_name(<-IMAN_reference<-ItemRevis
ion)
Test,A,Mfg0ArcOverrideForm,CustomForm_Name
```

As you can see, **Form** has the tag **Mfg0ArcOverrideForm**. This tag is mandatory, and differentiates custom forms from **Item** and **ItemRevision** master forms.

The GSID of the form is its parent's GSID plus an attribute value. Designate this information in the header as follows.

```
Form[Mfg0ArcOverrideForm]:object_name(<-IMAN_reference<-ItemRevision)
```

In this example, the form is attached to the **ItemRevision**. **ItemRevision** is its parent object. The value of **object_name** is used to compose the GSID. For example:

```
Form[Mfg0ArcOverrideForm]:object_desc(<-IMAN_reference<-Item)
```

In this example, the form's GSID is the **Item** GSID plus the **object_desc** value.

Dataset

Dataset has three related reference attributes: **ref_list**, **ref_names**, and **ref_types**. Manually populating these attributes in the CSV file can be complex, but the **csv2tcxml** utility can be used to attempt to set these attributes.

If both **Dataset** and **ImanFile** are in the header, they are linked automatically. For example:

```
Dataset:object_name,Dataset:dataset_type,ImanFile:original_file_name
DS_CSV_000001,UGMASTER,C:\Temp\ug.prt
```

The reference name for the **ImanFile** is deduced from the file extension. A file type can be referenced using different names. If the deduced reference name is undesirable, you can override the value by adding an additional **Dataset:ref_names** column.

Dataset can contain objects other than **ImanFile**, such as forms. To add additional attachments to a specific type of dataset, define parameters in the **DatasetAttachments** section of the *csv2tcxml.ini* file as follows:

```
UGPART=UGPartAttr:UGPART-ATTR,UGPartAttributesForm:UGPART-ATTRIBUTES
```

UGPART is the dataset type. Its value is a list of object type and reference name pairs. In this example, **UGPartAttr** is a Form type name and **UGPART-ATTR** is the reference name. Multiple attachments can be defined for one dataset by separating value pairs using commas. Attachments can be any type.

Once additional attachments are defined for a dataset, whenever an object of that dataset type is created, additional objects are constructed and linked to the dataset object.

You do not need to populate **ref_list** and its related attributes in the CSV input file, as the **csv2tcxml** utility automatically populates those values.

When overriding **ref_names** or **ref_types**, note that the forms **defined in DatasetAttachments** are set to the **ref_list** attribute before any **ImanFile**.

Link Dataset to ItemRevision

Dataset typically belongs to an **ItemRevision**. Using the **csv2tcxml** utility, you can achieve this relationship using **the Linking rule in the header**. For example:

```
Item:item_id,ItemRevision:item_revision_id,Dataset:dataset_type,Dataset:o
bject_name(<-IMAN_specification<-ItemRevision),
Part_010,A,UGMASTER,Part_010_DS
```

The approach in the example is recommended. However, this relationship can also be established using a separate **ImanRelation** column as follows.

```
Item:item_id,ItemRevision:item_revision_id,Dataset:dataset_type,Dataset:o
bject_name,ImanRelation:relation_type
Item_010,A,UGMASTER,DS_010,IMAN_specification
```

Work with Iman files

tcxml_import imports data from a file in the following manners:

- Import the physical file along with its metadata (the **ImanFile** object).
- Manually copy the physical file to a Teamcenter volume and import the **ImanFile** (without importing the physical file).

The first approach is suitable for importing relatively small amounts of data. The second approach is much faster when importing relatively large amounts of data.

Importing the physical file along with its metadata

When using this approach to import a physical file along with the **ImanFile** object, preparing the CSV file is straight forward. One attribute is needed:

```
ImanFile:original_file_name
C:\tmp\000113.prt
```

When **tcxml_import** runs, the physical file is copied from the provided location to the volume. If the physical file cannot be found, the import fails.

> Note:
>
> Do not supply the **file_name** column. **file_name** is automatically set during import.

ImanFile objects have two more mandatory attributes: **volume_tag** specifying in which volume this file should reside and **sd_path_name** specifying a subdirectory in the volume root.

Siemens Digital Industries Software recommends letting the **csv2tcxml** utility set these two attributes when the physical files are loaded when **tcxml_import** runs. Doing so requires that you provide Teamcenter administrator credentials when installing the **csv2tcxml** utility. These credentials allow the utility to retrieve volume information from Teamcenter. With this information, the utility determines the proper volume and directory based on the owning user and owning group.

Copying the file to a Teamcenter volume and importing only the ImanFile

Use the following steps for this approach.

1. Determine the volume and root subfolder in which to import the data.

2. Manually copy all physical files to this subfolder of the volume's root. File names must be unique in the folder.

3. In the **InitialValues** section in the initialization file, set **volume** to the volume name and set **sd_path_name** to the name of the subfolder. See **Customizing the csv2tcxml utility** for details.

4. In the CSV file, provide two columns:

   ```
   ImanFile:original_file_name,ImanFile:file_name
   000113.prt,000113.prt
   ```

   file_name is the name of the file on the disk. This file is created by the source system and is unique. If using musltiple source systems, ensure the file names are unique. This file name may be complex.

For example, *efbe000_unk_r4l0oc27obxlz.prt*. *original_file_name* is a human-readable name. No paths are needed for these files.

BOM structure

The BOM structure represented in the CSV file may have two styles: a PS occurrence (**PSOccurrence**) structure or an absolute occurrence (**AbsOccurrence**) structure.

Create a structure in either of the following manners:

- Convert and import the items. Then, construct another CSV for the BOM structure.
- Use the **csv2tcxml** utility to generate the items and BOM structure together in one conversion.

PSOccurrence structure

Following is an example of a **PSOccurrence** structure:

```
parent_item,parent_rev,child_item,child_rev
Top,A,Comp,A
Comp,A,Part,A
Top,A,Comp,A
Comp,A,Part,A
```

By default, the **csv2tcxml** utility creates precise assemblies. However, you can create imprecise assemblies by setting the **bvr_precise** parameter to a value of **0**. The default value for **bvr_precise** is **1**.

Define the type of BOM view and BOM view revision to create using the **bv_type** and **bvr_type** parameters. Set those parameters as needed, ensuring the base types match. By default, the values are **BOMView** and **BOMView Revision** respectively.

For **PSOccurrence** structures, each CSV line generates one **PSOccurrence** object linking the parent and child. In the earlier example, four lines are given, creating four **PSOccurrence** objects: two **PSOccurrence** objects for **Top/A** and **Comp/A** and two objects for **Comp/A** and **Part/A**. When opening the structure in Structure Manager, the following structure is shown:

```
Top/A
  Comp/A
    Part/A
    Part/A
  Comp/A
    Part/A
    Part/A
```

Note that there are 6 BOM lines under **Top/A** instead of 4. **Comp/A** occurs twice and each **Comp/A** has **Part/A** occurring twice. This structure is quite different than the **AbsOccurrence** structure.

If you provide the **PSOccurrence.qty_value** column, you can set the quantity value of the **PSOccurrence**. Only one **PSOccurrence** object is still created and you cannot unpack it. However, you can create multiple **PSOccurrence** objects by setting the **unpacked_psocc** parameter to a value of **1**. By setting **unpack_psocc** to **1**, multiple **PSOccurrence** objects are created, each with **qty_value** set to **-1**.

As you already create an **Item** through another CSV file, set **the exist parameter** to a value of **Item**. Updated **Item** and **Item Revision** objects are created in the output XML file. Some other objects, such as **Master Forms** and **Anchor** are not included.

Although importing a BOM in two steps is easy and straight-forward, you can import in a single step. Converting the same BOM CSV without setting **exist** to a value of **Item** and then importing the TCXML gives you a full BOM structure.

The column names used in the example are aliases. For clarity, consider using more appropriate names:

```
Item[Parent],ItemRevision[Parent],Item[Child],ItemRevision[Child]
```

All the names are fixed. Do not change the column names, as the following functions will no longer work.

```
Design[Parent],
Design_0_Revision_alt[Parent],Item[Child],ItemRevision[Child]
```

You can give an extra object_type column to use custom types. For example:

```
Item[Parent]:item_id,ItemRevision[Parent]:item_revision_id,Item[Parent]:object_type,Item[
Child]:item_id,ItemRevision[Child]:item_revision_id,Item[Child]:object_type
```

> Note:
>
> **object_type** is mandatory for all item types except **Item**.

For MFK items, ensure all of the MFK attributes are given in the BOM CSV file. For example, for an MFK Item defined as **{item_id,object_name}**, the BOM CSV header should be as follows:

```
Item[Parent]:item_id,Item[Parent]:object_name,ItemRevision[Parent]:item_revision_id,Item[
Child]:item_id,Item[Child]:object_name,ItemRevision[Child]:item_revision_id
```

AbsOccurrence structure

Following is an example of a **AbsOccurrence** structure similar to the earlier **PSOccurrence** structure:

```
parent_item,parent_rev,child_item,child_rev,apr_item
Top,A,Comp,A
Comp,A,Part,A
Top,A,Comp,A
Comp,A,Part,A
```

One additional; column is given to indicate that it is an **AbsOccurrence** structure. When opening the structure in Structure Manager, the following structure is shown:

```
Top/A
  Comp/A
    Part/A
  Comp/A
     Part/A
```

Note that there are four BOM lines under **Top/A**. **Comp/A** occurs twice and each **Comp/A** has **Part/A** occurring once.

In the CSV file, each line represents the appearance of one BOM line instead of a **PSOccurrence** object.

apr_item is used to designate the **MEAppearancePathRoot** for the specific line. If a valid ancestor item is given, **APR** (appearance path root), **APN** (appearance path node), and **AbsOccurrence** objects are created along the path.

> Note:
>
> When describing the BOM structure in an **AbsOccurrence** structure **CSV** file, complete an assembly before moving to its sibling assembly. Consider the following invalid example,
>
> ```
> parent_item,parent_rev,child_item,child_rev,apr_item
> Top,A,Comp,A
> Top,A,Comp,A
> Comp,A,Part,A
> ```
>
> As **Comp/A** is mentioned in the first data line, the **Comp/A** structure must then be described before continuing, similar to a depth-first tree traversal.

Lines can be omitted if the assembly has already been described.

```
Top/A
  Comp1/A
    Part1/A
    Part2/A
  Comp1/A
    Part1/A
    Part2/A
  Comp2/A
```

The following CSV input can be used to derive this structure:

```
parent_item,parent_rev,child_item,child_rev,apr_item
Top,A,Comp1,A
Comp1,A,Part1,A
Comp1,A,Part2,A
Top,A,Comp1,A
Comp1,A,Part1,A (optional)
Comp1,A,Part2,A (optional)
Top,A,Comp2,A
```

Since **Comp1/A** is completed in the first appearance under **Top**, you can remove the two optional lines and the result will be the same. If you need to create an **APN** node for a line, you cannot omit it and its siblings.

Revisable Occurrence / Usage structures

To create a structure consisting of **Revisable Occurrence** / **Usage** with multiple revisions, first create **Usage Product Item** and **Part Item** objects with a CSV file input similar to the following:

```
Item:item_id,ItemRevision:item_revision_id,Item:object_type,owning_user,owning_group,Item
:object_name;ItemRevision:object_name
BP010-0001,A,Sf9ProductItem,chenji,Engineering,Top Product
BP010-0002,A,Sf9PartItem,chenji,Engineering,Part 2
BP010-0003,A,Sf9PartItem,chenji,Engineering,Part 3
```

Overwrite the *model/datamodel/local_override.xml* file. Add the changes from *TC_DATA/csv2tcxml/examples/usagebom/sf9test_local_override.xml* to this file and rerun **the install steps**. The overrides take effect.

After completing the overrides, create a CSV input file as follows to create multiple **Revisable Occurrence** / **Usage** revisions as the input file:

```
parent_item,parent_rev,parent_type,PSOccurrence:child_item,child_rev,child_type,PSOccurre
nce:seq_no,PSOccurrence:qty_value,Ebm0PartUsageRevision:object_desc,PSOccurrence:occurren
ce_type(parent_bvr->Ebm0PartUsageRevision;occ_thread->PSOccurrenceThread),PSOccurrenceThr
ead:clone_stable_occ_uid,PSOccurrenceThread:gsid_suffix,Ebm0PartUsageRevision:object_type
,Ebm0PartUsageRevision:object_name,Ebm0PartUsageRevision:fnd0RevisionId(wso_thread->Ebm0P
artUsage),Ebm0PartUsage:fnd0BOMView,Ebm0PartUsage:fnd0ThreadId,PSBOMView:fnd0StructManage
mentMode,Ebm0OccAttrs:ebm0StockDisposition(fnd0OwningPSOcc->PSOccurrence)
BP010-0001,A,Ebm0PartProduct,BP010-0002!
Part,A,Part,10,-1,sample_desc,0,AAAAAAAAAAAAAA,gsid1,Ebm0PartUsageRevision,obj_name_wsot,
A,BP010-0001!Ebm0PartProduct!BOMView,11111,1,DISP
BP010-0001,A,Ebm0PartProduct,BP010-0002!
Part,A,Part,10,-1,sample_desc,0,AAAAAAAAAAAAAA,gsid1,Ebm0PartUsageRevision,obj_name_wsot,
B,BP010-0001!Ebm0PartProduct!BOMView,11111,1,NOST
BP010-0001,A,Ebm0PartProduct,BP010-0003!
Part,A,Part,20,-1,sample_desc,0,AAAAAAAAAAAAAA,gsid2,Ebm0PartUsageRevision,obj_name_wsot,
A,BP010-0001!Ebm0PartProduct!BOMView,12121,1,DISP
BP010-0001,A,Ebm0PartProduct,BP010-0003!
Part,A,Part,20,-1,sample_desc,0,AAAAAAAAAAAAAA,gsid2,Ebm0PartUsageRevision,obj_name_wsot,
B,BP010-0001!Ebm0PartProduct!BOMView,12121,1,RWK
BP010-0001,A,Ebm0PartProduct,BP010-0003!
Part,A,Part,20,-1,sample_desc,0,AAAAAAAAAAAAAA,gsid2,Ebm0PartUsageRevision,obj_name_wsot,
C,BP010-0001!Ebm0PartProduct!BOMView,12121,1,
```

After successfully completing the conversion, revert the added changes from *model/datamodel/local_override.xml* and rerun the installation steps.

GDE lines in BOM structures

To create a structure with generic design element (GDE) lines, use a header with the following form:

```
Item[Parent]:item_id,Item[Parent]:object_type,ItemRevision[Parent]:item_revision_id,Item[
Child]:item_id,Item[Child]:object_type,ItemRevision[Child]:item_revision_id,GeneralDesign
Element[Child]:object_type,GeneralDesignElement[Child]:object_name
```

GeneralDesignElement is a child of the **Form** business object. Use a dynamic type through **object_type** for both **Item** and **GeneralDesignElement**.

In the header, note that both parent and child items are provided similar to in typical BOM structure CSV files. **GeneralDesignElement[Child]** is added for a GDE line under the BOM line. Use **Item[Parent]** to designate the parent line (instead of using **Item[Child]**). If no child item BOM line exists, **Item[Child]** columns can be removed.

If a GDE line has its own child GDE lines, provide **GeneralDesignElement[Parent]** and use it with **GeneralDesignElement[Child]** to create a parent-child relationship. Refer to the following example: *TC_DATA\csv2tcxml\examples\foundation\GDEStructure.csv*.

Refer to *TC_DATA\csv2tcxml\examples\foundation\GDEStructureWithConnection.csv* to create a GDE structure with a connection as in the following example:

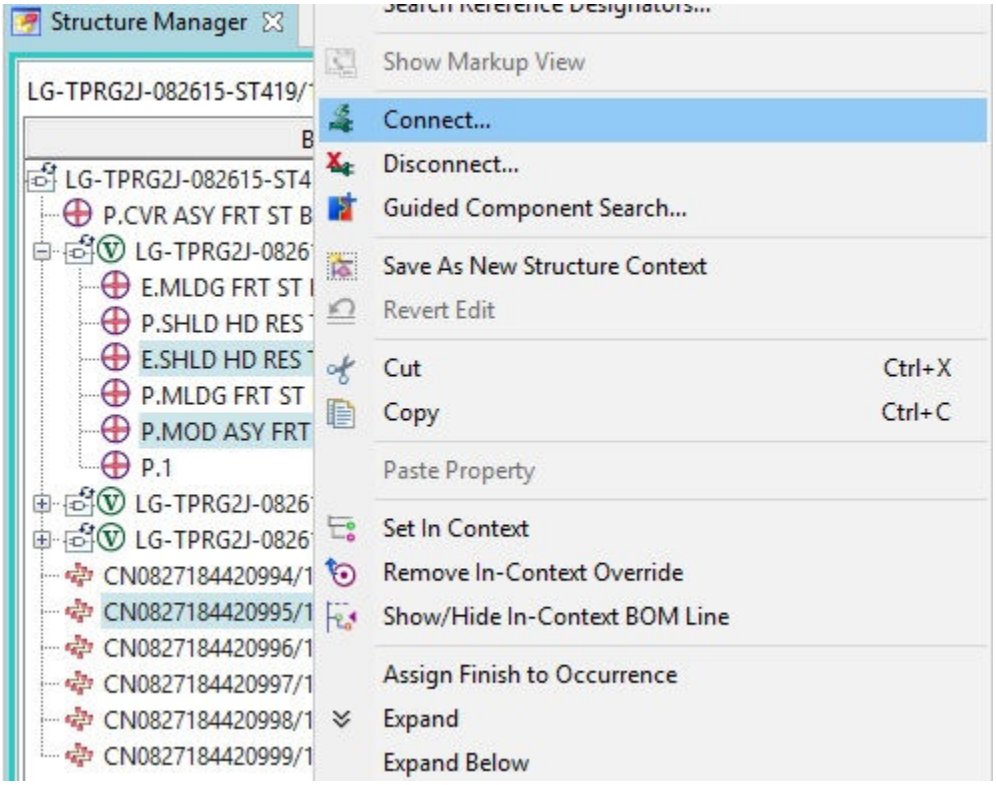


Table property

Each table property contains one or more table rows which are subclasses of **Fnd0TableRow**. Before creating the table property, determine which object contains which type of table row and what the property name is.

In the following example, the property owner is omitted:

```
Mfg0MEConnectorTableRow:fnd0OwningPropName(fnd0OwningObject->PropertyOwner),Mfg0MEConnect
orTableRow:fnd0RowIndex,Mfg0MEConnectorTableRow:mfg0ConnectorID,Mfg0MEConnectorTableRow:m
fg0ConnectorName
mfg0ConnectorTable,"0,1,2","ID001,ID002,ID003","Con1,Con2,Con3"
mfg0ConnectorTable,"0,1","ID004,ID005","Con4,Con5"
```

The first column is the property name. The second is the row index. The index always starts from **0**. The third and fourth columns are row values. Note that **Fnd0TableRow** is defined as an aggregation class. Because of this definition, you can provide multiple values in one column. Multiple table rows are created.

4GD support

The *csv2tcxml* utility has the following 4GD support.

Cpd0CollaborationDesign

For example:

mdl0model_id and **object_name** are mandatory for 4GD objects. For example:

```
Cpd0CollaborativeDesign:mdl0model_id,Cpd0CollaborativeDesign:object_name
CD00001,MyCD
```

In this example, the **object_name** of **CD** is "MyCD".

Partitions and their hierarchy

Use the following form to create a partition structure:

```
Cpd0CollaborativeDesign:mdl0model_id,Ptn0Partition[Parent]:ptn0partition_
id,Ptn0Partition[Parent]:object_type,Ptn0Partition[Child]:ptn0partition_i
d,Ptn0Partition[Child]:object_type
```

All of the columns are mandatory for **Partition** structures.

- Every partition belongs to a **CD**, so you need to provide the **CD mdl0model_id**.
- **Ptn0Partition[Parent]:ptn0partition_id,Ptn0Partition[Parent]:object_type** provides the parent partition information, **Ptn0Partition[Child]:ptn0partition_id,Ptn0Partition[Child]:object_type** is for the child partition. This form is similar to a BOM structure where you provide tags to identify the **Parent** and **Child**.
- You cannot change the column names for **Ptn0Partition** as with a BOM structure. The names are fixed.

If you set parameter **exist=Cpd0CollaborativeDesign,Ptn0Partition**, the output will contain only **Ptn0ChildParentLink**. This approach lets you first import **Partition** and **CD** before constructing the partition hierarchy later.

If the **CD** is already created by this utility and you later want to construct the partition structure for it using the parameter **exist=Cpd0CollaborativeDesign**, first ensure the proper partition schemas are already created for the **CD**. With **exist=Cpd0CollaborativeDesign**, creating the partition schema is skipped.

After importing the partitions, run the following command to refresh the partition cache to see the partitions.

```
populate_top_level_partitions_cache -model=<model_id>
```

Promissory and shape design elements

Creating Promissory and Shape design elements is more straight-forwards than creating reuse and subordinate design elements as no structure is involved.

Promissory design elements	`Cpd0CollaborativeDesign:mdl0model_id,Cpd0DesignElement:cpd0design_element_id` No **cpd0category** is given, as the default, the design element category is "promissory". For promissory design elements, provide one id column for **CD** and one id column for the design element.
Shape design elements	`Cpd0CollaborativeDesign:mdl0model_id,Cpd0DesignElement:cpd0design_element_id,Cpd0DesignElement:cpd0category,Cpd0ShapeDesign:item_id,Cpd0ShapeDesignRevision:item_revision_id` For shape design elements, provide **cpd0category**. One shape design **Item** also is needed for a shape design element. The two are linked automatically.

Reuse and subordinate design elements

Reuse and subordinate design elements are created together in a single CSV file:

```
Cpd0CollaborativeDesign:mdl0model_id,Cpd0DesignElement:cpd0design_element_id,Cpd0DesignElement:cpd0category,Cpd0DesignElement:cpd0is_leaf,Item:item_id,ItemRevision:item_revision_id,Cpd0DesignElement[Parent]:cpd0design_element_id,Item[Parent]:item_id,ItemRevision[Parent]:item_revision_id,Cpd0DesignElement[Top]:cpd0design_element_id
```

The **DesignElement** value without a tag describes the design element you are creating. The **DesignElement** with a tag (**Parent**) provides the parent information for the current design element. If a **Parent** is empty, it is a top or reuse design element.

If a design element has a parent, it is a subordinate design element. Subordinate design elements require parent and top design element information using the **Cpd0DesignElement[Top]:cpd0design_element_id** column. Besides design elements, Items are also needed. Every reuse or subordinate design element has a corresponding Item.

Provide the design element structure from top to bottom. Every parent design element must appear before its children. Refer to the samples in *TC_DATA\csv2tcxml\examples*.

AttributeGroup and ManagedAttributeGroup

For example:

```
Cpd0CollaborativeDesign:mdl0model_id,Ptn0Design:ptn0partition_id,Cpd0Desi
gnElement:cpd0design_element_id,Cpd0DesignElement:cpd0category,Cpd0InfoIn
Context:object_name(<-Mdl0AttachAttrGroup<-Cpd0DesignElement),Cpd0InfoInC
ontext:cpd0FindNum
```

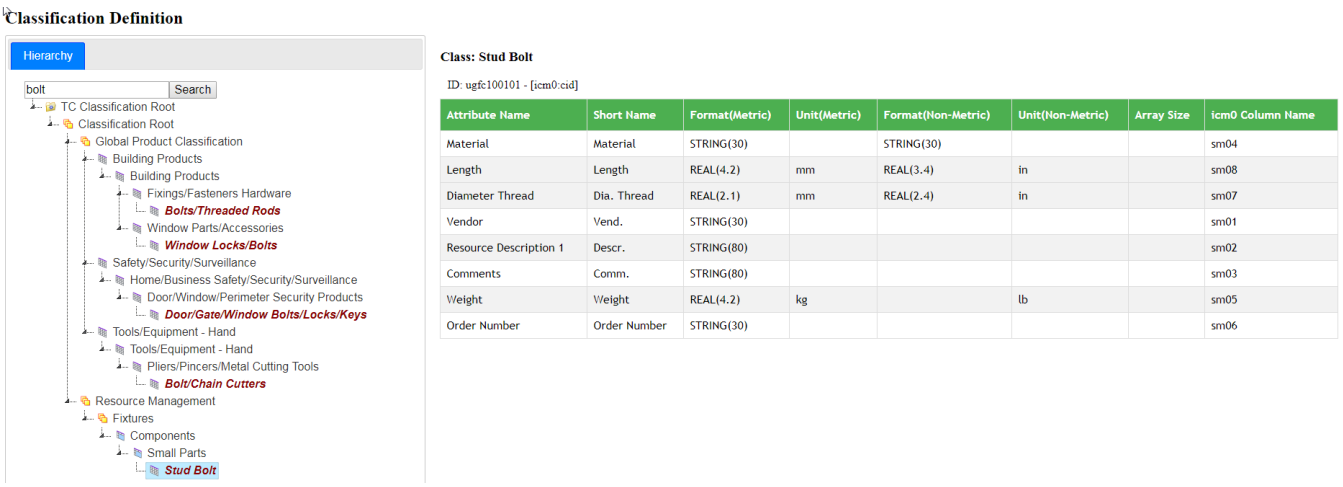
Cpd0InfoInContext is an **Mdl0AttributeGroup** attribute group. It does not have MFK or **keyAttributes** values defined. **Cpd0InfoInContext** can be linked to a design element using a header linker rule to create and link the object.

The **ImanRelation** type is **Mdl0AttachAttrGroup**. In the earlier example, the GSID of **Cpd0InfoInContext** is the GSID of the design element + **Cpd0InfoInContext:object_name** + "**Cpd0InfoInContext**".

Classification object

To create classification objects (ICOs), you must construct an **icm0** object in the Teamcenter database. Since ICOs of different classes share the same table, **icm0** has a long list of general attributes – **sm01** to **sm200**. When you set attribute values, you must know which of these columns to use.

To identify the column name, refer to *TC_DATA\csv2tcxml\model\classification.html*. (Refer to **Install and set up the csv2tcxml utility** for steps on generating this file.) Following is an example of the information contained in the file.



The class hierarchy is shown as a tree in which you can search by class name. Once you select a class, the attributes for that class are displayed, including a column identifying where to put the attribute value for an ICO object (**icm0**).

If an attribute is an array (and **Array Size** is not empty), surround the value with curly braces even though it has only one specific value.

Every **icm0** object sets the **cid** attribute to the class ID, shown just below the class name on *classification.html*.

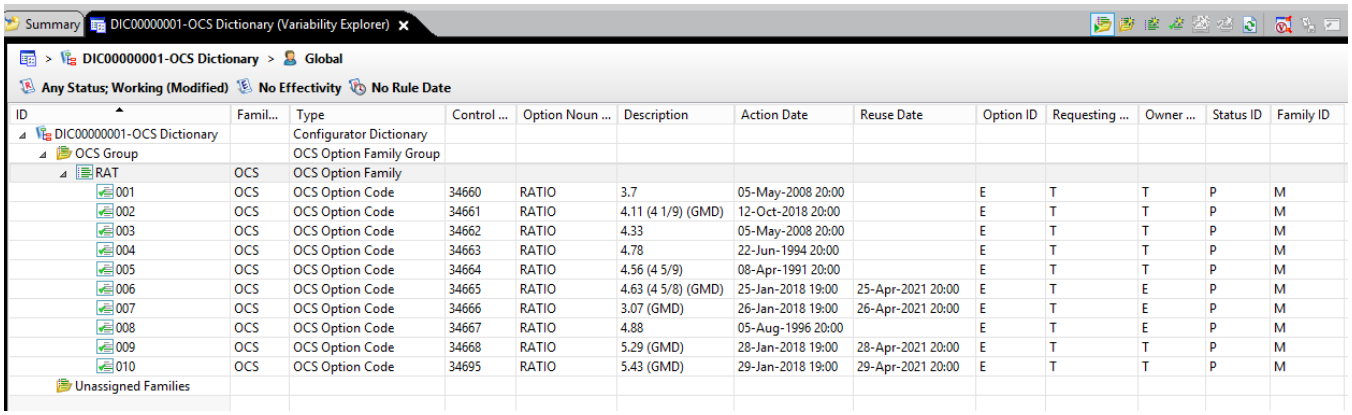
There are five types of attributes:

Attribute Type	Description
String	**STRING(MAX_LENGTH)**. Value doesn't needs to be quoted.
Integer	**INTEGER(MAX_DIGITS)**
Float Number	**REAL(X.Y)**. If the format shows REAL(2.4), value should be like 10.0000.
Date	**DATE(YYMMDD)**. Then you need to give date value like 20190215.
Key LOV	The page shows all the possible key/value pairs. Provide the key as the value.

System of Measurement		• m:Metric • nm:Non-Metric					sm4
Base unit		• Y:Yes • N:No					sm5

Product configurator

You can use the **csv2tcxml** utility to migrate product configurator structures. For example:



ID	Famil...	Type	Control ...	Option Noun ...	Description	Action Date	Reuse Date	Option ID	Requesting ...	Owner ...	Status ID	Family ID
DIC00000001-OCS Dictionary		Configurator Dictionary										
OCS Group		OCS Option Family Group										
RAT	OCS	OCS Option Family										
001	OCS	OCS Option Code	34660	RATIO	3.7	05-May-2008 20:00		E	T	T	P	M
002	OCS	OCS Option Code	34661	RATIO	4.11 (4 1/9) (GMD)	12-Oct-2018 20:00		E	T	T	P	M
003	OCS	OCS Option Code	34662	RATIO	4.33	05-May-2008 20:00		E	T	T	P	M
004	OCS	OCS Option Code	34663	RATIO	4.78	22-Jun-1994 20:00		E	T	T	P	M
005	OCS	OCS Option Code	34664	RATIO	4.56 (4 5/9)	08-Apr-1991 20:00		E	T	T	P	M
006	OCS	OCS Option Code	34665	RATIO	4.63 (4 5/8) (GMD)	25-Jan-2018 19:00	25-Apr-2021 20:00	E	T	E	P	M
007	OCS	OCS Option Code	34666	RATIO	3.07 (GMD)	26-Jan-2018 19:00	26-Apr-2021 20:00	E	T	E	P	M
008	OCS	OCS Option Code	34667	RATIO	4.88	05-Aug-1996 20:00		E	T	E	P	M
009	OCS	OCS Option Code	34668	RATIO	5.29 (GMD)	28-Jan-2018 19:00	28-Apr-2021 20:00	E	T	T	P	M
010	OCS	OCS Option Code	34695	RATIO	5.43 (GMD)	29-Jan-2018 19:00	29-Apr-2021 20:00	E	T	T	P	M
Unassigned Families												

Use the following steps to construct the structure in this example:

1. Create **Product Item – DIC00000001** as in the example.
2. Create **Feature** and **Feature Family**.
3. Link **Feature** and **Feature Family** using **Cfg0FeatureFamily:cfg0FeatureThreads**.
4. Create **Family Group**.
5. Link **Family Group** and **Feature Family** using **Cfg0FamilyGroup:cfg0FamilyThreads**.

Refer to the examples in *TC_DATA\csv2tcxml\examples\configurator*.

If the **Feature** and **Feature Family** are **Cfg0LiteralOptionValue** and **Cfg0LiteralValueFamily** respectively, use the following steps to accommodate the data model differences.

1. Create **Product Item**.
2. Create **Cfg0LiteralValueFamily**.
3. Create **Cfg0LiteralOptionValue**. Provide **Cfg0LiteralFamilyThread:fnd0ThreadId** to link **Cfg0LiteralOptionValue** to **Feature Family**.
4. Create **Family Group**. Link **Family Group** and **Feature Family** using **Cfg0FamilyGroup:cfg0FamilyThreads**.

Schedule and ScheduleTask

Schedule and **ScheduleTask** both have **item_id** as their key attribute.

Create a **Schedule** structure as follows:

1. Create the **Schedule** object and its associated **TCCalender**. Refer to the examples in *TC_DATA\csv2tcxml\examples\foundation\Schedule.csv*.

2. Create **ScheduleTasks**. As **ScheduleTask** has attributes pointing to parent and children tasks, provide a GSID string for the reference. Refer to the examples in *TC_DATA\csv2tcxml\examples\foundation\ScheduleTask.csv*.

3. Create a dependency relation between tasks. *TC_DATA\csv2tcxml\examples\foundation\ScheduleTaskDependency.csv*.

Be aware that you can convert all of the objects in a single run. Use the following command and then load the structure.

```
csv2tcxml.bat Schedule.csv ScheduleTask.csv ScheduleTaskDependency.csv
-o Schedule.xml
```

Update mode

Update mode overview

Once you have created data in the Teamcenter database, you may find that you need amend the data by updating attributes. You can do so with the **csv2tcxml** utility by setting the **mode parameter** to a value of **update**.

Setting the **mode** parameter to **update** updates the values of attributes specified in the CSV file. Attributes not provided in the CSV file (aside from **lsd** and **last_mod_date**) are not updated.

Objects must already exist in the database. If they do not, **tcxml_import** fails when attempting to update an attribute value.

> Note:
>
> Setting the **mode** parameter causes the utility to operate differently than when setting the **exist** parameter. Setting **exist** causes the utility to skip object creation for objects already in the database.

Identify the object

To update an object, you must first identify the object using one of the following values:

- PUID. Directly provide a PUID to identify an object. Objects which are not created by this utility tool can only be identified using their PUID.

  ```
  ItemRevision:object_desc,ItemRevision:puid
  ItemRevision description for CSV_00001/A,eIWXeGRTuTyVvA
  ```

  puid is a custom attribute valid only for update mode.

- GSID. Provide the GSID string for an object as in the following example.

```
ItemRevision:gsid,ItemRevision:object_desc
CSV_00001!Item!A!ItemRevision,ItemRevision description for CSV_00001/A
```

  gsid is a custom attribute valid only for update mode. Siemens Digital Industries Software recommends using the PUID to identify an object whenever possible.

- Key attribute. Only principal objects created with the **csv2tcxml** utility can use this method.

```
Item:item_id,Item:object_desc,Item:owning_user
CSV_00001,Item created by csv2tcxml,chenji
```

  In this example, **item_id** is the key attribute used to identify the Item.

> Caution:
>
> Modifying key attribute values may cause conversion errors that are highly difficult to diagnose. Siemens Digital Industries Software recommends creating new objects instead of modifying key attributes.

The methods support the following objects:

Identify object by	Principal objects created by csv2tcxml	Helper objects created by csv2tcxml	Objects not created by csv2tcxml
PUID	X	X	X
GSID	X	X	
Key attributes	X		

Update object attributes

Once an object is identified, you can add extra columns to modify its attributes.

- No helper object is created.
- No extension method is called.
- No initial values from are populated from the data model.
- No linking rules and object links are established.

Provide the value you want to set for an attribute. If that attribute is a reference, provide the PUID or GSID. It the attribute is an admin reference, provide a candidate key value. The **csv2tcxml** utility **validates values** according to the attribute type.

During import, the **tcxml_import** utility compares the **lsd** of the object in the database and **lsd** of the element in the TC XML. The update succeeds only when the **lsd** in the TCXML is more recent. By default, the **csv2tcxml** utility sets the **lsd** to the current time. If the object is a **POM_application_object**, **last_mod_date** is also set to the same date as **lsd**. To keep **last_mod_date** unchanged, set the **update_lmd_disabled parameter** to a value of **1**.

> Note:
>
> Certain attributes, such as **project_list**, **ip_classification**, and **gov_classification**, cannot be updated because they have application logic bound to them.

Extending the CSV data model

Overview of extending the data model

You can modify the CSV data model file to extend the **csv2tcxml** utility capabilities. Extensions can include:

- Change initial values for attributes.
- Define **keyAttribute** values for a class.
- Define helper for a class.
- Define **propagateAttribute** values for a class.
- Set **isCandidateKey** to define the class as an admin class.

Siemens Digital Industries Software does not recommend modifying the original *csv2tcxml_datamodel.xml* file. Instead, update the *\csv2tcxml\model\datamodel\local_override.xml* file to override the settings in *csv2tcxml_datamodel.xml*. Add your changes to *local_override.xml* and rerun the **utility installation steps** for the changes to take effect.

You can override only **TcClass** and its **TcAttribute**.

Supporting a new object type

Use the following steps to add support for new objects.

1. Determine if the new object is a principal object or a helper object. See **Principal object and its helper** for information on crating helper objects with principal objects. Also see **Custom forms** for information on creating help objects using custom forms.

2. If the object needs to be created alone, that is, without linking it to some other object, create it as a principal object. Confirm it is not already a principal object by reviewing *TC_DATA\csv2tcxml\model\datamodel\datamodel.html* to verify the object does not have a key definition.

3. If the object does not have a key defined, try to determine a combination of attributes that can uniquely identify the object. If you can find no such combination, you can provide a custom attribute such as **gsid_suffix** as a key. Ensure that the **gsid_suffix** value is unique and provide this value in the CSV file.

   Include your key definition in the model file in the *TC_DATA\csv2tcxml\model\datamodel* folder. If no data model file exists in the folder for your template, create a new file in the *datamodel* directory for your template. For example, if your template file is *abc_template.xml*, create a data model file named *abc.xml* in the folder.

   Use the following example from *foundation.xml* as a guide:

   ```
   <TcClass className="Fnd0AdminLOVValue"
   keyAttributes="object_name,fnd0lov_category">
   </TcClass>
   ```

4. Create your CSV input file, providing at least one key attribute, and convert the file.

   If the conversion fails with errors such as "Mandatory attributes are missing" or "Required properties are missing", provide initial values for those attributes.

   Define initial values as follows:

   ```
   <TcClass className="ClsName" keyAttributes="your key definition">
   <TcAttribute attributeName="attrName" initialValue="your value"/>
   ...
   </TcClass>
   ```

5. Rerun **the utility installation steps** to consolidate your customization in the data model.

Be aware that if you make you changes in the consolidated csv2tcxml data model file (*TC_DATA\csv2tcxml\model\datamodel\csv2tcxml_datamodel.xml*), the changes are lost when re-running the install step.

Supported encoding standards

By default, the *csv2tcxml* utility processes CSV files as UTF-8 encoded files. The *csv2tcxml* utility supports several CSV file encoding standards in addition to UTF-8. To use files with other encodings, specify the encoding standard using the encoding value in the *csv2tcxml.ini* file. TC XML output files are always encoded as UTF-8.

The *csv2tcxml* utility supports the following encoding standards.

7bit-jis	cp875	MacArabic
AdobeStandardEncoding	cp932	MacCentralEurRoman
AdobeSymbol	cp936	MacChineseSimp

AdobeZdingbat
ascii
ascii-ctrl
big5-eten
big5-hkscs
cp1006
cp1026
cp1047
cp1250
cp1251
cp1252
cp1253
cp1254
cp1255
cp1256
cp1257
cp1258
cp37
cp424
cp437
cp500
cp737
cp775
cp850
cp852
cp855
cp856
cp857
cp858
cp860
cp861
cp862
cp863
cp864
cp865
cp866
cp949
cp950
dingbats
euc-cn
euc-jp
euc-kr
gb12345-raw
gb2312-raw
gsm0338
hp-roman8
hz
iso-2022-jp
iso-2022-jp-1
iso-2022-kr
iso-8859-1
iso-8859-10
iso-8859-11
iso-8859-13
iso-8859-14
iso-8859-15
iso-8859-16
iso-8859-2
iso-8859-3
iso-8859-4
iso-8859-5
iso-8859-6
iso-8859-7
iso-8859-8
iso-8859-9
iso-ir-165
jis0201-raw
jis0208-raw
jis0212-raw
johab
koi8-f
koi8-r
MacChineseTrad
MacCroatian
MacCyrillic
MacDingbats
MacFarsi
MacGreek
MacHebrew
MacIcelandic
MacJapanese
MacKorean
MacRoman
MacRomanian
MacRumanian
MacSami
MacSymbol
MacThai
MacTurkish
MacUkrainian
MIME-B
MIME-Header
MIME-Header-ISO_2022_JP MIME-Q
nextstep
null
posix-bc
shiftjis
symbol
UCS-2BE
UCS-2LE
UTF-16
UTF-16BE
UTF-16LE
UTF-32
UTF-32BE
UTF-32LE
UTF-7
utf-8-strict

cp869	koi8-u	utf8
cp874	ksc5601-raw	viscii

Customizing the csv2tcxml utility

The csv2tcxml utility is configurable. You can change its default behavior through parameters. All parameters are defined in sections of the *csv2tcxml.ini* file. For example, **encoding** is defined in the **Config** section and **sep** is defined in **CsvReader** section. Parameter names are case-sensitive.

Avoid changing values in the *csv2tcxml.ini* file. Instead, create a dedicated *.ini* file for your data. It is recommended that you do not change any parameters in *csv2tcxml.ini* except for **source_site** in the **[Config]** section.

For example, if the file *utf8_encoding.csv* uses utf8 encoding. Create a file named *utf8_encoding.csv.ini* and add the following content to it:

```
[Config]
encoding=utf8
```

For another file, *gb2312_encoding.csv* which uses gb2312 encoding, create a separate file named *gb2312_encoding.csv.ini*set **encoding** to **gb2312** in its **Config** section. This approach modularizes the configuration and is less prone to error.

Parameters in an *.ini* file can also be overridden by the command line option **–p**. This approach is helpful for automation. If you are setting a parameter through the command line for the **[Config]** section, you don't need to provide the section name. Overriding parameters from any other section *csv2tcxml.ini* must include the name of the section. For example, to override the value of the **sep** parameter in the **CsvReader** section, use the form **–p [CsvReader]sep=#**.

The *csv2tcxml.ini* file contains the following sections an parameters.

Config

Name	Description	Default Value
source_site	A site ID representing an external site or system. This parameter must be set before using the **csv2tcxml** utility. Contact your Siemens Digital Industries Software representative to for site IDs.	N/A
encoding	The input file encoding type. See **Supported encoding standards** for a list of valid file encoding values.	**ascii**
db_encoding	The database encoding mapped to the **encoding** value. By default, the **csv2tcxml** utility uses **utf-8** encoding to determine if a string value exceeds the maximum string length limit. If your database is using a specific encoding	**utf-8**

Name	Description	Default Value
	(for example, shiftjis), set **db_encoding** to that encoding value to ensure so the **csv2tcxml** utility processes values properly.	
GMS_tcxml_string_separator	The separator used in TC XML to split VLA (variable length array) attribute values.	**,** (comma)
exist	Skip creation of objects that already exist in the database. The value is the object name or the class name in the header. Helper objects are also skipped.	N/A
grouping	Combine different lines that generate the same object of the class defined with this parameter. Provide only one class name and an optional tag name such as **Parent**. Class hierarchy is supported.	N/A
save_gsid_out	Save the PUID/GSID mapping to a file when set to **1**.	**0**
puid_lookup_file	Specify the file path of a text file containing a list of existing PUID/GSID pairs of objects not created by the *csv2tcxml.ini* utility.	N/A
Island_batch_size	Specify the number of objects to be grouped in an island. • Objects generated for one line are always in the same island. • To have each line be an island, set this value to **-1**. • To import all lines without creating islands, set this value to **0**.	**6000**
create_form_storage	To create a storage object for every Form object, set this parameter to **1**. By default, the *csv2tcxml.ini* utility does not create a storage class if the storage class is not specified in the *.csv* file.	**0**
skip_empty_value	Set this parameter to a column name to skip empty values in that column. The utility sets an initial value for that attribute if there is an initial value, otherwise the attribute is not created for the specific object.	N/A
skip_line_if_error	Set this parameter to **1** to skip the entire line if any error occurs for the line. By default, partial or faulty data is created for the line to facilitate error diagnoses. Correct any errors before importing the output XML file.	N/A
local_time_zone	In TC XML, the time is always GMT. If a local time string is given, set this parameter to the time zone and the utility converts the local time to GMT. For example, **-500** specifies the UTC-05:00 time zone; **GMT** specifies GMT. If this parameter is not set, the utility	N/A

Name	Description	Default Value
	only recognizes time specified using the following format: **\d{4}\D\d{2}\D\d{2}\D\d{2}\D\d{2}\D\d{2}**.	
lov_validate	By default, the utility validates LOV (list of values) values. Disable this validation by setting this option to **0**.	**1**
bvr_precise	Specify if the utility should create a precise or imprecise structure. Set this parameter to **1** for precise or to **0** for imprecise.	**1**
bv_type	The type of BOM view for BOM creation.	**BOMView**
bvr_type	The type of BOM view revision for BOM creation.	**BOMView Revision**
unpacked_psocc	By default, a single **PSOccurrence** is created and **qty_value** is directly set. To create multiple **PSOccurrence** objects if **PSOccurrence.qty_value** is more than **1**, set this parameter to **1**.	**0**
mode	Specify the conversion mode. A value of **Full** causes the utility to perform a full conversion. A value of **Update** causes the utility to use update mode.	**Full**
update_lmd_disabled	Specifies if the last modified date is updated upon conversion. When set to **1**, the utility does not update the last modified date. **update_lmd_disabled** is used when converting using update mode.	**0**

InitialValues

This section contains no fixed parameters. You can add or update initial values for any class. Parameters in this section have the following form:

```
ClassName=Attribute1:Value1,Attribute2:Value2
```

For example, set the following parameter to specify an initial value for **POM_application_object.owning_user**. Doing so avoids setting the same value in every *.csv* file.

```
POM_application_object=owning_user:migration_user
```

CsvReader

Name	Description	Default Value
sep	Specifies the characters used to separate fields. The separator can be one or more characters, but is limited to 8 bytes. For example, ",", "~", and "~##" are all valid values. Ensure the header and data use the same separator.	**, | -auto detect**
quote	Specifies the characters used to quote fields. The separator can be one or more characters, but is limited to 8 bytes.	**"**
escape_char	Specifies the character to escape certain characters inside quoted fields. The character must be a single-byte character, typically in the range of 0x20 (space) to 0x7E (tilde).	**"**
strict	When this parameter is set to **1**, any row that parses to a different number of fields than the header causes the utility to report an error and that row is ignored. An empty row is treated as error.	**0**
binary	When this parameter is set to **1**, binary characters (including line feeds, carriage returns, and NULL bytes) may be used in quoted fields. NULL bytes can be escaped as **"0**.	**1**

QueryDB

Parameters for querying an object's PUID from the database using attributes.

Name	Description	Default Value
query_db	The classes to be queried from the database to find the object's PUID. The value is the class name shown in the header. Multiple class names are separated by commas.	N/A
tc_admin_user	Specifies the user ID. The user must have administrative privileges.	N/A
tc_admin_password	Specifies the user password. **tc_admin_password** is mutually exclusive with **tc_admin_password_file**.	N/A
tc_admin_password_file	Specifies the password file. **tc_admin_password_file** is mutually exclusive with **tc_admin_password**.	N/A

DatasetAttachments

Parameters in this section are optional. Set the parameters as needed.

Name	Description	Default Value
dataset_class	Defines any additional objects to create when the specified dataset is created.	**UGPartAttr:UGP ART- ATTR,**

Name	Description	Default Value
	The parameter name *dataset_class* is any valid Teamcenter dataset type. For example, **UGPart** or **PDF**. The value is a comma-separated list of additional attachments to create with the dataset using the following format: `TypeName1:ReferenceName1,TypeName2:ReferenceName2`	**UGPartAttributesForm: UGPA RT-ATTRIBUTES**

PhysicalStructure

Parameters specifying whether to create an as-built structure or an as-maintained structure.

Name	Description	Default Value
isAsBuiltStructure	If **isAsBuiltStructure** is set to **0**, an as-maintained structure is created. If **isAsBuiltStructure** is set to **1**, an as-built structure is created.	**0**

PropagationRule

Parameters to define value list ordering.

Name	Description	Default Value
ITAR_level_list_ordering	Defines ITAR clearance and classification levels. See Defining ITAR clearance and classification levels.	**secret super-secret,top-secret**
IP_level_list_ordering	Defines IP clearance and classification levels. See Defining IP clearance and classification levels.	**secret super-secret,top-secret**

Other undocumented sections

Additional parameter sections and parameters may be included in *csv2tcxml.ini*. Those parameters are for internal use. Do not change them.

Admin classes and candidate attributes

When a class has a candidate attribute defined in BMIDE, it is considered an admin class. Following are common admin classes and their candidate attributes. This is not a complete list, as new candidate keys may be defined for any class.

Class Name	Candidate Key Attribute
ADA_License	id
Condition	condition_name
DatasetType	datasettype_name
Discipline	discipline_name
Group	full_name (A runtime property.)
ImanType	type_name
ImanVolume	Volume_name
NoteType	name
POM_imc	site_id
PSOccurrenceType	name
PSViewType	name
RevisionRule	object_name
Role	role_name
TC_Project	project_id
Tool	object_name
UnitOfMeasure	symbol

Note that candidate key definitions are inherited by child classes.

Supported classes

The following parent classes are supported. Child classes are automatically supported (in absence of custom application logic).

Foundation

AbsOccData	Form	ProjectObjectRelation
AbsOccurrence	GDEbvr	PSBOMView
Anchor	GDEOccurrence	PSBOMViewRevision
Dataset	GeneralDesignElement	PSOccurrence
Effectivity	icm0	PSOccurrenceNotes
Fnd0AdminLOVValue	ImanFile	PSOccurrenceThread
Fnd0LogicalBlock	ImanRelation	ReleaseStatus
Fnd0LogicConn	Item	RevisionAnchor
Fnd0OARFunction	ItemRevision	Schedule

Fnd0OARRule	MEAppearancePathNode	ScheduleTask
Fnd0TableRow	MEAppearancePathRoot	

Appmodel

Mdl0AttachAttrGroup	Mdl0DefaultGeometry	Mdl0ManagedAttrGroup
Mdl0AttributeGroup	Mdl0ElementThread	

Cpd

Cpd0CollaborativeDesign	Cpd0ShapeDesign	Cpd0DesignItemInstance
Cpd0DesignElement	Cpd0ShapeDesignRevision	

Partition

Ptn0ChildParentLink	Ptn0PartitionItem	Ptn0SchemeFunctional
Ptn0Membership	Ptn0PartitionItemRevision	
Ptn0Partition	Ptn0PartitionTemplateModel	

Realization

Rlz0ItemRealizationMap

Mdc0mdconnectivity

Mdc0ConnectionElement	Mdc0OrderedElementGroup
Mdc0ConditionalElementGroup	Mdc0PortArtifact

Smd0systemmodeling

Smd0LogicalElement	Smd0Model

Pdm0plantdatamgmt

Pdm0ExternalModel	Pdm0ImportRecord
Pdm0ExternalType	Pdm0RelatedModel

Cfg0configurator

Cfg0FamilyGroup	**Cfg0FeatureFamily**	**Cfg0LiteralValueFamily**
Cfg0Feature	**Cfg0LiteralOptionValue**	**Cfg0ProductItem**

Mrocore

AsBuiltStructure	**PhysicalLocationRevision**	**PhysicalPartRevision**
AsMaintainedStructure	**PhysicalLocationUsage**	**PhysicalRealization**
PhysicalLocation	**PhysicalPart**	**PhysicalStructureAlignment**

Serviceplanning

SSP0ServicePartition	**SSP0ServiceReq**
SSP0ServicePlan	**SSP0WorkCard**

Preparing Teamcenter for bulk data import

Enable bulk loader utilities

Fast import of objects using the **tcxml_import** utility uses data files that conform to the low-level (LL) TC XML format. The **bulk load process** also uses the LL TC XML format. The use of stub objects and inferred delete are handled differently in this format.

For LL TC XML imports, an entire island of data must be full elements regardless of whether an object is locally owned or a replica. Stub objects are used to solve island interdependencies and are represented by **POM_stub** objects in the Teamcenter database. They are not used to represent objects that have already been sent to the target site as it does in HL TC XML imports. Consequently, if you transfer a new revision to Teamcenter using LL TC XML, all predecessor revisions must be transferred as full objects. These may have further related objects and so on. If they are not full objects, an inferred delete of the stub objects occurs.

Bulk load Teamcenter data

From the perspective of the **Teamcenter components**, you use the following process to load bulk data into Teamcenter:

1. Use the APIs **TIE_get_preconstructed_uids** and **TIE_get_hashed_uid** to generate Teamcenter UIDs for composing the TC XML file containing legacy data.

```
int TIE_get_preconstructed_uids
(
   int   numRequestedUids,  (I)  Number of UIDs to construct
```

```
    int* numConstructedUids, (O)  Number of UIDs constructed
    char*** constructedUids  (OF) Number of numConstructedUids
                                  constructed
);

int TIE_get_hashed_uid
(
  int ownSiteId,       (I)  Site Id of Legacy system
  const char* hashKey, (I)  Hash Key
  char** hashedUID     (OF) Valid Teamcenter UID for given hash key
);
```

2. Use the **tcxml_validate** utility to validate the TC XML file generated by the custom mapping solution. The validator log reports potential errors within TC XML file. Fix the root cause of any errors within the custom mapping logic. This validation focuses on data samples (~2500 element XML files) extracted from legacy systems to be executed in a test lab environment.

3. Use the **tcxml_import** utility to import the TC XML files generated by the custom mapping solution. After all aspects of the translation/mapping are correct, the full TC XML file can imported at the highest speed possible, without any additional validation.

   Caution:

   The importing user must have write access to the location where the utility logs the information about the import. You can ensure that this requirement is met by including the **-log** argument with a path to a location that is write accessible to the importing user. You may optionally include a file name for the log file in the path.

   When attribute values contain commas in the strings at either of the sites, you must change the default value for the **GMS_tcxml_string_separator** preference. Change this value at both sites to prevent data corruption at the importing site.

Tip:

The view type data is not exported with the **PSBOMView** data in TC XML. The import process uses the default view type when importing **PSBOMView** data.

Therefore, when transferring **PSBOMView** data with any utility that uses TC XML metadata, set the **PSE_default_view_type** preference at the target site to the view type value of the assembly at the source site. Usually, this is the same as the source sites default view type. Briefcase files, the **tcxml_export** utility, and the **tcxml_import** utility each uses TC XML metadata.

This import behavior is designed to be consistent with Structure Manager.

For reference information useful for bulk loader customizations, see the following topics:

- *Teamcenter core data dictionary*
- *Teamcenter data model diagram*

- *Sample assembly structure in a TC XML file*
- *Sample TC XML file with GSID references*
- *Sample bulk load repeat file*
- *Frequently asked questions about bulk data loading*

TC XML file with GSID references

Bulk loader supports importing low-level (LL) TC XML files containing global stable ID (GSID) references (**GSIdentity** elements). GSIDs are used for object look-up and the bulk load import handles creation of Teamcenter UIDs for the legacy objects. This relieves the custom mapping solution from invoking a Teamcenter API for UID creation and identity management.

Caution:

You must set the **GMS_USE_FNV_HASH** preference to **true** to enable the use of the FNV hashing API to create Teamcenter UIDs for the legacy objects.

Note:

If you use the bulk load function to load objects that will be referenced when loading data, the **GSIdentity** element must match the element used by the data exchange process.

Every element in a TC XML file containing GSID references:

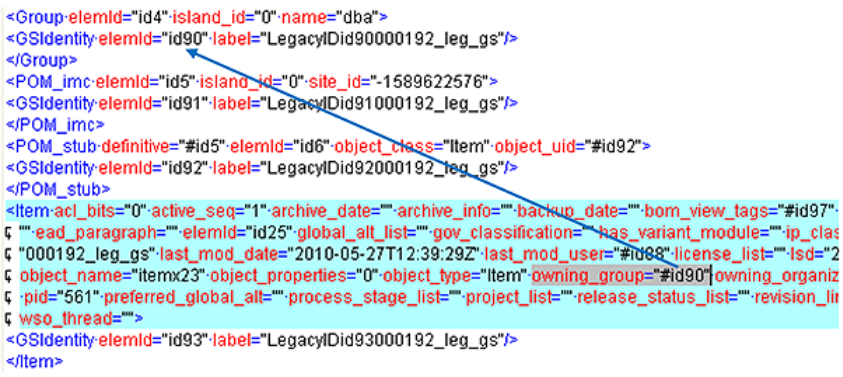
- Uses the leaf-class name as the element name.
- Is uniquely identified by its **elemId** attribute (required).
- Has an **island_id** attribute (required).
- Has a **GSIdentity** element as child element (required) with:
  - An **elemId** attribute that uniquely identifies the element (required).
  - A label attribute with a unique value for each element (required).

The XML elements for organizational (**User**, **Group**, **Site**, and so forth) and administrative (**Types**, **Rules**) objects:

- Are assigned **0** for the **island_id** attribute
- Contain the candidate key attribute

In a GSID-based TC XML file, the organizational objects are referred by the child **GSIdentity** element's **elemId** attribute of the corresponding organization object. For example, the **owning_group** attribute of the **Item** element in the following figure, points to the child **GSIdentity** element's **elemId** attribute of the **Group** object.

Following is a typical GSID-based organization object reference:



Each object may contain an associated child **GSIdentity** element and the object look - up is based on the **GSIdentity** element. The **puid** attribute may or may not be present and, in any case, the **GSIdentity** element takes precedence for identifying the object. For new objects, the **GSIdentity** attributes (**label**, **subLabel**, and **split_token**) are used to generate the **puid** attribute. The **GSIdentity** elements for the new objects are persisted when the island is saved. They serve as import records for these objects. Bulk load import also allows mixed TC XML files where some objects have UID-based identity and some have GSID-based identity.

Reference attributes refer to other objects by the **elemId** attribute of the child **GSIdentity** element. The following example shows this type of reference.

In the example, the **items_tag** attribute of the **ItemRevision** element has a value of **#id93**. This value matches the **elemId** attribute of the **Item** object's child **GSIdentity** element.

TC XML file with POM_stub elements

In Teamcenter, **POM_stub** objects represent objects that do not exist in the database. These objects are always locally owned and are the database objects that represent a stubbed object.

The **POM_stub** element requires three attributes:

- **definitive**

  Points to the source site **POM_imc** element in the TC XML file.

- **object_class**

  Indicates the class of the object represented by the stub.

- **object_uid**

  Points to the **GSIdentity** element of the object represented by the stub. If the actual object is not present in the TC XML file, the **object_uid** refers to its own **GSIdentity** element uniquely identifies the object as being represented by the stub. In this case, the attributes of the **GSIdentity** element of the **POM_stub** element are used to compose the **object_uid** attribute.

An example of a TC XML file with POM stubs:

If the import process detects that the actual object exists in the Teamcenter database for the stub, it skips creation of the stub.

Create, update, and infer-delete

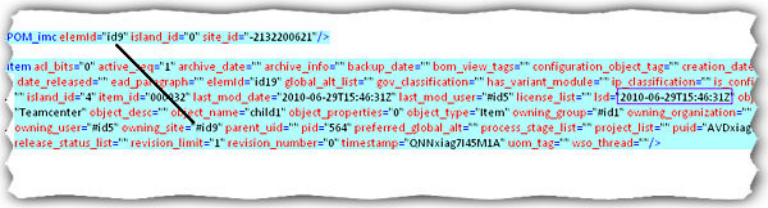
The import functionality of the bulk load process is different from low-level fast imports in that some objects are imported as replicas followed by a separate step for ownership transfer. Also the objects owned by the target site are not updated. During bulk load, the objects are imported as local objects at the target Teamcenter site and are updated in all subsequent imports. The **bulk_load** session option allows for this variation in behavior. This option is set when the **-bulk_load** argument is used with the **tcxml_import** utility.

For creation, the **owning_site** attribute of the object must point to a **POM_imc** element that contains the **site-id** attribute of the importing Teamcenter site in the TC XML file.

For updates, the **lsd** attribute of the object in the TC XML file must be later than the **lsd** attribute of the object in the Teamcenter database.

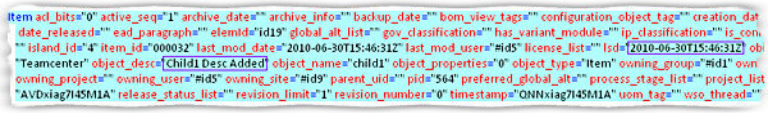
The fast import traverses the target database to collect the helpers of each primary object in the TC XML file. The objects that exist in the target database but not in the TC XML file are deleted if they are not locally owned. This is the infer-delete mechanism. Because the bulk load process imports objects as local to Teamcenter, the infer-delete logic does not apply.

The following examples show the create, update, and skip sequence of the import process:



The bulk load process creates an item with an **item_id** value of **000032** in Teamcenter site **-2132200621** as a local object.

The bulk load process skips the item with an **item_id** value of **000032** because it already exists at Teamcenter site **-2132200621** and the **lsd** attribute value is the same in the TC XML file and the database.



The bulk load process updates the item with an **item_id** value of **000032** because it already exists at Teamcenter site **-2132200621** and the **lsd** attribute value in the TC XML file is later than the value in the database.

Teamcenter core data dictionary

The following tables show the mandatory attributes for the core Teamcenter classes (that is, those attributes that are to be present for each class instance in the TC XML file).

Note:

When using the bulk load utility (**tcxml_import**), the utility sets appropriate values for attributes that are not mandatory. This type of activity must be done for other classes of interest as part of the customization work.

Item attributes

Attribute	Mandatory	Comments
acl_bits	No	
active_seq	No	
archive_date	No	
archive_info	No	
backup_date	No	
configuration_object_tag	No	
creation_date	No	
date_released	No	
ead_paragraph	No	
elemId	Yes	
global_alt_list	No	
gov_classification	No	
has_variant_module	No	
ip_classification	No	
is_configuration_item	No	
is_vi	No	
island_id	Yes	
item_id	Yes	
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
license_list	No	
lsd	No	

Attribute	Mandatory	Comments
object_application	Yes	Typically set to **Teamcenter**.
object_desc	No	
object_name	Yes	
object_properties	No	
object_type	Yes	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_organization	No	
owning_project	No	
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
parent_uid	Yes	
pid	No	**Class-id**. Fast import ignores this attribute.
preferred_global_alt	No	
process_stage_list	No	
project_list	No	
puid	Yes	
release_status_list	No	
revision_limit	Yes	Typically set to **1**.
revision_number	Yes	Typically set to **0**.
timestamp	No	
uom_tag	No	
wso_thread	No	

Anchor elements

Attribute	Mandatory	Comments
acl_bits	No	
archive_date	No	
archive_info	No	
backup_date	No	
creation_date	No	
elemId	Yes	
immune_objects	No	
island_id	Yes	
keep_limit	Yes	Typically set to **3**.
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
lsd	No	
managed_objects	No	
object_properties	No	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_uid	Yes	
puid	Yes	

ItemRevision attributes

Attribute	Mandatory	Comments
acl_bits	No	
active_seq	No	
archive_date	No	

Attribute	Mandatory	Comments
archive_info	No	
backup_date	No	
creation_date	No	
date_released	No	
declared_options	No	
ead_paragraph	No	
elemId	Yes	
gde_bvr_list	No	
gov_classification	No	
has_variant_module	No	
ip_classification	No	
island_id	Yes	
item_revision_id	Yes	
items_tag	Yes	Holds the **PUID** value of the corresponding Item.
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
license_list	No	
lsd	No	
object_application	No	
object_desc	No	
object_name	Yes	
object_properties	No	
object_type	Yes	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_organization	No	
owning_project	No	

Attribute	Mandatory	Comments
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
parent_uid	Yes	
pid	Yes	The **class-id**. Fast import ignores the attribute.
process_stage_list	No	
project_list	No	
puid	Yes	
release_status_list	No	
revision_limit	Yes	Typically set to **1**.
revision_number	Yes	Typically set to **0**.
sequence_anchor	Yes	Holds the **PUID** value of the corresponding **Anchor** element.
sequence_id	Yes	Typically set to **1**.
sequence_limit	Yes	Typically set to **3**.
timestamp	No	
used_options	No	
variant_expression_block	No	
wso_thread	No	

Form attributes

Attribute	Mandatory	Comments
acl_bits	No	
active_seq	No	
archive_date	No	
archive_info	No	
backup_date	No	

Attribute	Mandatory	Comments
creation_date	No	
data_file	No	
date_released	No	
ead_paragraph	No	
elemId	Yes	
form_file	Yes	Typically set to **n/a**.
gov_classification	No	
ip_classification	No	
island_id	Yes	
last_mod_date	No	
last_mod_user		Points to the **User** element in the TC XML file.
license_list	No	
lsd	No	
object_application	Yes	Typically set to **Teamcenter**.
object_desc	No	
object_name	Yes	
object_properties	No	
object_type	Yes	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_organization	No	
owning_project	No	
owning_user	Yes	Points to the **User** element in the TC XML file.
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_uid	Yes	
pid	Yes	The **class-id**. Fast import ignores the attribute.

Attribute	Mandatory	Comments
process_stage_list	No	
project_list	No	
puid	Yes	
release_status_list	No	
revision_limit	Yes	Typically set to **1**.
revision_number	Yes	Typically set to **0**.
timestamp	No	
wso_thread	No	

ImanRelation attributes

Attribute	Mandatory	Comments
elemId	Yes	
island_id	Yes	
lsd	No	
object_properties	No	
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_uid	Yes	
pid	Yes	The **class-id** attribute. Fast Import ignores the attribute.
primary_object	Yes	Holds the **PUID** value of the primary object of the **ImanRelation** object.
puid	Yes	
relation_type	Yes	Points to the corresponding **ImanType** element in the TC XML file.
secondary_object	Yes	Holds the **PUID** value of the secondary object of the **ImanRelation** object.

Attribute	Mandatory	Comments
timestamp	No	
user_data	No	

PSBOMView attributes

Attribute	Mandatory	Comments
acl_bits	No	
active_seq	No	
archive_date	No	
archive_info	No	
backup_date	No	
creation_date	No	
date_released	No	
ead_paragraph	No	
elemId	Yes	
gov_classification	No	
ip_classification	No	
island_id	Yes	
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
license_list	No	
lsd	No	
object_application	Yes	Typically set to **Teamcenter**.
object_desc	No	
object_name	Yes	
object_properties	No	
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
object_type	Yes	

Attribute	Mandatory	Comments
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
parent_item	Yes	
parent_uid	Yes	
pid	No	The **class-id** attribute. Fast Import ignores the attribute.
process_stage_list	No	
project_list	No	
puid	Yes	
release_status_list	No	
revision_limit	Yes	Typically set to **1**.
revision_number	Yes	Typically set to **0**.
timestamp	No	
view_type	Yes	Points to the **PSViewType** element in the TC XML file.
wso_thread	No	

PSBOMViewRevision attributes

Attribute	Mandatory	Comments
acl_bits	No	
active_seq	No	
archive_date	No	
archive_info	No	
backup_date	No	
bom_view	Yes	Points the **PUID** value of the corresponding **PSBOMView** element.
creation_date	No	
date_released	No	

Attribute	Mandatory	Comments
ead_paragraph	No	
elemId	Yes	
gov_classification	No	
island_id	Yes	
is_precise	Yes	Typically set to **0**.
ip_classification	No	
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
license_list	No	
lsd	No	
object_application	Yes	Typically set to **Teamcenter**.
object_desc	No	
object_name	Yes	
object_properties	No	
object_type	Yes	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_organization	No	
owning_project	No	
owning_site	Yes	Required for LL TC XML. Points to the **POM_imc** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
parent_uid	Yes	
pid	Yes	
process_stage_list	No	
project_list	No	
puid	Yes	

Attribute	Mandatory	Comments
release_status_list	No	
revision_limit	Yes	Typically set to **1**.
revision_number	Yes	Typically set to **0**.
struct_last_mod_date	Yes	
timestamp	No	
wso_thread	No	

PSOccurrence attributes

Attribute	Mandatory	Comments
alternate_etc_ref	No	
child_item	Yes	Points the **PUID** value of the corresponding item.
child_bv	No	
effectivities	No	
elemId	Yes	
island_id	Yes	
lsd	No	
notes_ref	No	
object_properties	No	
occ_thread	Yes	Points the **PUID** value of the corresponding **PSOccurrenceThread** element in the TC XML file.
occ_flags	No	
occ_type	No	
occurrence_type	No	
order_no	No	Typically set to **10**.
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_bvr	Yes	Points the **PUID** value of the corresponding

Attribute	Mandatory	Comments
		PSBOMViewRevision element in the TC XML file.
parent_uid	Yes	Required for low-level TC XML schema.
pid	No	
pred_list	No	
puid	Yes	
qty_value	No	Typically set to **-1**.
seq_no	Yes	Typically set to **10**.
timestamp	No	
uom_tag	No	
used_options	No	
variant_condition	No	
xform	No	

PSOccurenceThread attributes

Attribute	Mandatory	Comments
clone_stable_occ_uid	No	
elemId	Yes	
island_id	Yes	
lsd	No	
object_properties	No	
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_uid	Yes	
pid	Yes	The **class-id** attribute. Fast import ignores this attribute.
puid	Yes	
timestamp	No	

Dataset attributes

Attribute	Mandatory	Comments
acl_bits	No	
active_seq	No	
archive_date	No	
archive_info	No	
backup_date	No	
creation_date	No	
dataset_type	Yes	Points to the **DatasetType** element in the TC XML file.
date_released	No	
ead_paragraph	No	
elemId	Yes	
format_used	No	
gov_classification	No	
ip_classification	No	
island_id	Yes	
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
license_list	No	
local_path	No	
lsd	No	
markup_acl	No	
markup_create_tool	No	
markup_official	No	
markup_status	No	
object_application	Yes	Typically set to **Teamcenter**.
object_desc	No	
object_name	Yes	

Attribute	Mandatory	Comments
object_properties	No	
object_type	Yes	
owning_organization	No	
owning_project	No	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
parent_uid	Yes	
pid	No	
process_stage_list	No	
project_list	No	
puid	Yes	
ref_names	No	
ref_list	Yes	Holds the **PUID** value of the corresponding **ImanFile** element in the TC XML file.
ref_types	No	
release_status_list	No	
rev_chain_anchor	Yes	Holds the **PUID** value of the corresponding **RevisionAnchor** element in the TC XML file.
revision_limit	Yes	Typically set to **1**.
revision_number	Yes	Typically set to **0**.
system_managed	No	
tool_used	Yes	Points to the **Tool** element in the TC XML file.
timestamp	No	

Attribute	Mandatory	Comments
user_class	No	
wso_thread	No	

RevisionAnchor attributes

Attribute	Mandatory	Comments
acl_bits	No	
archive_date	No	
archive_info	No	
backup_date	No	
creation_date	No	
elemId	Yes	
highest_rev	Yes	Typically set to **1**.
id	No	
island_id	Yes	
keep_limit	Yes	Typically set to **3**.
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
lsd	No	
object_properties	No	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_uid	Yes	
pid	Yes	
puid	Yes	
rev	No	

Attribute	Mandatory	Comments
revisions	Yes	Holds the values of the corresponding **Dataset** revisions.
timestamp	No	

ImanFile attributes

Attribute	Mandatory	Comments
acl_bits	No	
archive_date	No	
archive_info	No	
backup_date	No	
creation_date	No	
destination_volume_tag	No	
elemId	Yes	
file_name	Yes	
hsm_info	No	
island_id	Yes	
last_mod_date	No	
last_mod_user	Yes	Points to the **User** element in the TC XML file.
lsd	No	
machine_type	No	
object_properties	No	
original_file_name	No	
owning_group	Yes	Points to the **Group** element in the TC XML file.
owning_user	Yes	Points to the **User** element in the TC XML file.
owning_site	Yes	Points to the **POM_imc** element in the TC XML file.
parent_uid	Yes	

Attribute	Mandatory	Comments
pid	Yes	The **class-id** attribute. Fast import ignores this attribute.
puid	Yes	
relative_directory_path	No	
released_version	No	
sd_path_name	Yes	Contains the relative path of the legacy file from the logical volume root.
status_flag	No	
store_and_forward_flag	No	
text_flag	No	
time_last_modified	No	
timestamp	No	
translate	No	
volume_tag	Yes	Points to the **ImanVolume** element in the TC XML file.
vm_info	No	

User attributes

Attribute	Mandatory	Comments
elemId	Yes	
island_id	Yes	
user_id	Yes	Candidate key

Group attributes

Attribute	Mandatory	Comments
elemId	Yes	
island_id	Yes	
name	Yes	Candidate key

Tool attributes

Attribute	Mandatory	Comments
elemId	Yes	
island_id	Yes	
object_name	Yes	Candidate key

ImanType attributes

Attribute	Mandatory	Comments
elemId	Yes	
island_id	Yes	
type_class	Yes	Candidate key
type_name	Yes	Candidate key

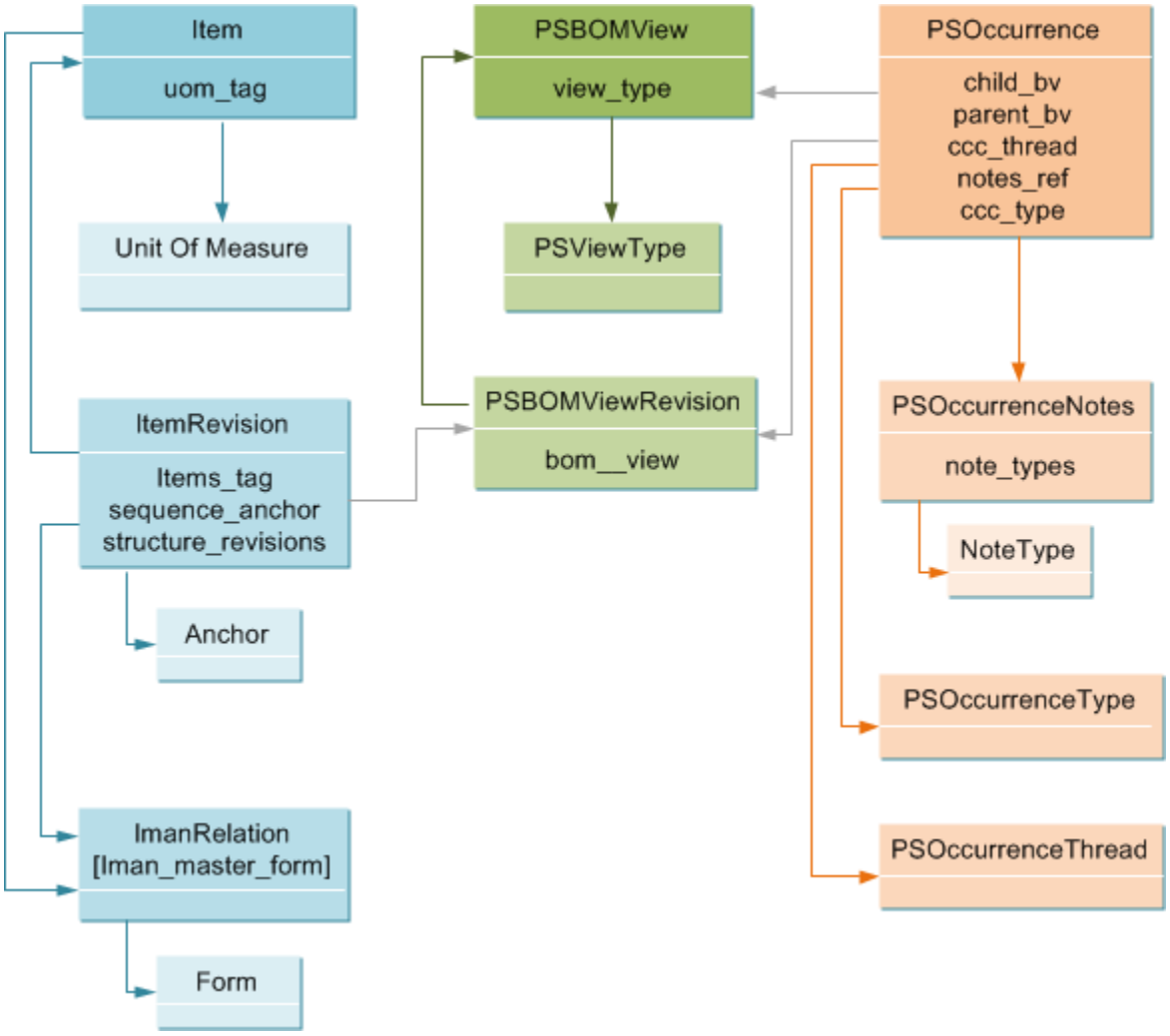
ImanVolume attributes

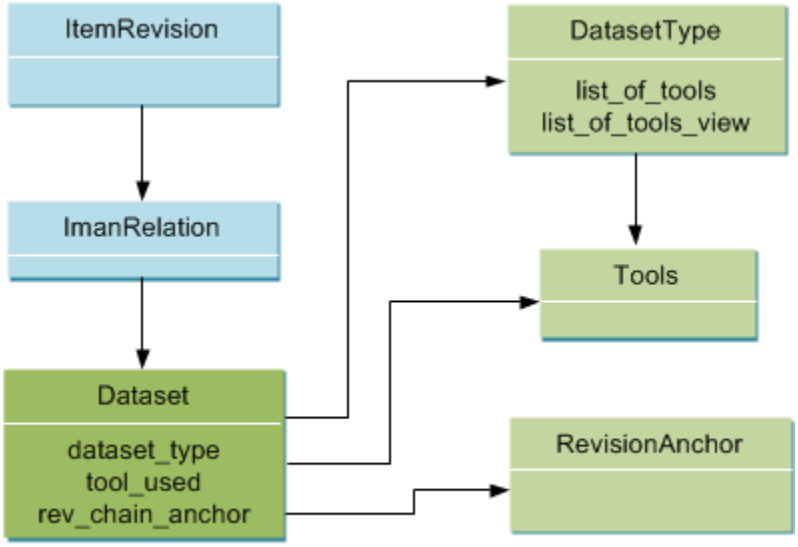
Attribute	Mandatory	Comments
elemId	Yes	
island_id	Yes	
volume_name	Yes	Candidate key

DatasetType attributes

Attribute	Mandatory	Comments
elemId	Yes	
datasettype_name	Yes	Candidate key
island_id	Yes	

Teamcenter data model diagram

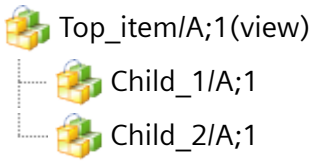




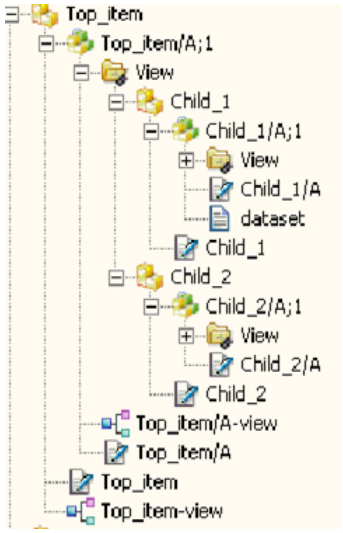
Sample assembly structure in a TC XML file

The structure in the following figure is a simple one-level bill of material (BOM) structure where:

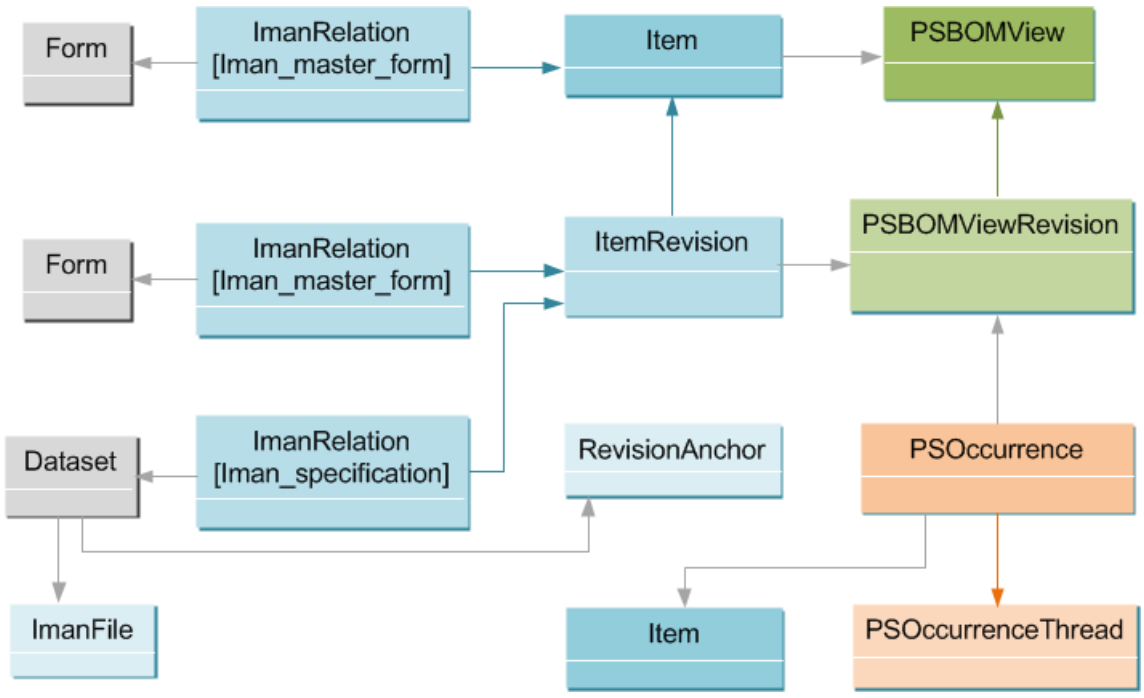
- The **Top_item** object is the parent item.
- The **Child_1** and **Child_2** objects are the children of the **Top_item** object.
- A dataset is attached to the **Child_1** object.



The expanded view of the **Top_item** object and its children includes the **dataset** object attached to the **Child_1** object:



The following figure shows the relationships between the objects.



Sample TC XML file with GSID references

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <TCXML xmlns="http://www.tcxml.org/Schemas/TCXMLSchema">
- <Group elemId="id1" island_id="0" name="dba">
  <GSIdentity elemId="id75" label="LegacyIDid75one" />
  </Group>
- <ImanType elemId="id2" island_id="0" type_class="ImanRelation"
type_name="IMAN_master_form">
  <GSIdentity elemId="id76" label="LegacyIDid76one" />
  </ImanType>
```

```
- <User elemId="id3" island_id="0" user_id="tcdba">
  <GSIdentity elemId="id77" label="LegacyIDid77one" />
  </User>
- <PSViewType elemId="id4" island_id="0" name="view">
  <GSIdentity elemId="id78" label="LegacyIDid78one" />
  </PSViewType>
+ <POM_imc elemId="id5" island_id="0" site_id="-1589622576">
+ <ItemRevision acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:07Z" date_released="" declared_options=""
ead_paragraph="" elemId="id6"
    gde_bvr_list="" gov_classification="" has_variant_module="" ip_classification=""
island_id="5"
   item_revision_id="A" items_tag="#id81" last_mod_date="2021-05-29T10:23:07Z"
last_mod_user="#id77"
   license_list="" lsd="2021-05-29T10:23:07Z" object_application="Teamcenter" object_desc=""
   object_name="onec" object_properties="0" object_type="ItemRevision" owning_group="#id75"
   owning_organization="" owning_project="" owning_user="#id77" parent_uid="#id81" pid="758"
   process_stage_list="" project_list="" release_status_list="" revision_limit="1"
revision_number="0"
   sequence_anchor="#id82" sequence_id="1" sequence_limit="3" timestamp="QpHx6UvxI45M1A"
used_options=""
   variant_expression_block="" wso_thread="">
- <Item acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
configuration_object_tag=""
   creation_date="2021-05-29T10:23:06Z" date_released="" ead_paragraph="" elemId="id7"
global_alt_list=""
   gov_classification="" has_variant_module="" ip_classification="" is_configuration_item=""
is_vi=""
   island_id="5" item_id="onec" last_mod_date="2021-05-29T10:23:07Z" last_mod_user="#id77"
license_list=""
   lsd="2021-05-29T10:23:07Z" object_application="Teamcenter" object_desc=""
object_name="onec"
   object_properties="0" object_type="Item" owning_group="#id75" owning_organization=""
owning_project=""
   owning_user="#id77" parent_uid="" pid="564" preferred_global_alt="" process_stage_list=""
project_list=""
   release_status_list="" revision_limit="1" revision_number="0" timestamp="QpIx6UvxI45M1A"
uom_tag=""
   wso_thread="">
   <GSIdentity elemId="id81" label="LegacyIDid81one" />
  </Item>
- <Anchor acl_bits="0" archive_date="" archive_info="" backup_date=""
creation_date="2021-05-29T10:23:07Z"
   elemId="id8" immune_objects="" island_id="5" keep_limit="3"
last_mod_date="2021-05-29T10:23:07Z"
   last_mod_user="#id77" lsd="2021-05-29T10:23:07Z" managed_objects="" object_properties="0"
   owning_group="#id75" owning_user="#id77" parent_uid="#id80" pid="520"
timestamp="QlMx6UvxI45M1A">
   <GSIdentity elemId="id82" label="LegacyIDid82one" />
  </Anchor>
- <ImanRelation elemId="id9" island_id="5" lsd="2021-05-29T10:23:06Z" object_properties="0"
parent_uid="#id81"
   pid="443" primary_object="#id81" relation_type="#id76" secondary_object="#id85"
timestamp="QhGx6UvxI45M1A"
   user_data="">
   <GSIdentity elemId="id83" label="LegacyIDid83one" />
  </ImanRelation>
- <ImanRelation elemId="id10" island_id="5" lsd="2021-05-29T10:23:07Z" object_properties="0"
parent_uid="#id80"
   pid="443" primary_object="#id80" relation_type="#id76" secondary_object="#id86"
```

```
timestamp="QpEx6UvxI45M1A"
   user_data="">
   <GSIdentity elemId="id84" label="LegacyIDid84one" />
  </ImanRelation>
- <Form acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:06Z" data_file="" date_released="" ead_paragraph=""
elemId="id11"
   form_file="n/a" gov_classification="" ip_classification="" island_id="5"
   last_mod_date="2021-05-29T10:23:06Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:23:06Z"
   object_application="Teamcenter" object_desc="" object_name="onec" object_properties="0"
   object_type="Item Master" owning_group="#id75" owning_organization="" owning_project=""
owning_user="#id77"
   parent_uid="#id83" pid="555" process_stage_list="" project_list="" release_status_list=""
   revision_limit="1" revision_number="0" timestamp="QdEx6UvxI45M1A" wso_thread="">
   <GSIdentity elemId="id85" label="LegacyIDid85one" />
  </Form>
- <Form acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:07Z" data_file="" date_released="" ead_paragraph=""
elemId="id12"
   form_file="n/a" gov_classification="" ip_classification="" island_id="5"
   last_mod_date="2021-05-29T10:23:07Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:23:07Z"
   object_application="Teamcenter" object_desc="" object_name="onec/A" object_properties="0"
   object_type="ItemRevision Master" owning_group="#id75" owning_organization=""
owning_project=""
   owning_user="#id77" parent_uid="#id84" pid="555" process_stage_list="" project_list=""
   release_status_list="" revision_limit="1" revision_number="0" timestamp="QpBx6UvxI45M1A"
wso_thread="">
  <GSIdentity elemId="id86" label="LegacyIDid86one" />
  </Form>
- <ItemRevision acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:35Z" date_released="" declared_options=""
ead_paragraph="" elemId="id13"
   gde_bvr_list="" gov_classification="" has_variant_module="" ip_classification=""
island_id="3"
   item_revision_id="A" items_tag="#id88" last_mod_date="2021-05-29T10:24:26Z"
last_mod_user="#id77"
   license_list="" lsd="2021-05-29T10:24:26Z" object_application="Teamcenter" object_desc=""
   object_name="one" object_properties="0" object_type="ItemRevision" owning_group="#id75"
   owning_organization="" owning_project="" owning_user="#id77" parent_uid="#id88" pid="758"
   process_stage_list="" project_list="" release_status_list="" revision_limit="1"
revision_number="0"
   sequence_anchor="#id93" sequence_id="1" sequence_limit="3" structure_revisions="#id92"
   timestamp="gBLx6UvxI45M1A" used_options="" variant_expression_block="" wso_thread="">
   <GSIdentity elemId="id87" label="LegacyIDid87one" />
  </ItemRevision>
- <Item acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
bom_view_tags="#id90"
   configuration_object_tag="" creation_date="2021-05-29T10:23:35Z" date_released=""
ead_paragraph=""
   elemId="id14" global_alt_list="" gov_classification="" has_variant_module=""
ip_classification=""
   is_configuration_item="" is_vi="" island_id="3" item_id="one"
last_mod_date="2021-05-29T10:24:26Z"
   last_mod_user="#id77" license_list="" lsd="2021-05-29T10:24:26Z"
object_application="Teamcenter"
   object_desc="" object_name="one" object_properties="0" object_type="Item"
owning_group="#id75"
   owning_organization="" owning_project="" owning_user="#id77" parent_uid="" pid="564"
```

```
   preferred_global_alt="" process_stage_list="" project_list="" release_status_list=""
revision_limit="1"
   revision_number="0" timestamp="gBMx6UvxI45M1A" uom_tag="" wso_thread="">
   <GSIdentity elemId="id88" label="LegacyIDid88one" />
  </Item>
- <PSOccurrence alternate_etc_ref="" child_bv="" child_item="#id81" effectivities=""
elemId="id15"
   island_id="3" lsd="2021-05-29T10:24:26Z" notes_ref="" object_properties="0" occ_flags="0"
   occ_thread="#id91" occ_type="" occurrence_type="0" order_no="10" parent_bvr="#id92"
parent_uid="#id92"
   pid="783" pred_list="" qty_value="-1" seq_no="10" timestamp="gBEx6UvxI45M1A" uom_tag=""
used_options=""
   variant_condition="" xform="">
   <GSIdentity elemId="id89" label="LegacyIDid89one" />
  </PSOccurrence>
- <PSBOMView acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:24:16Z" date_released="" ead_paragraph="" elemId="id16"
   gov_classification="" ip_classification="" island_id="3"
last_mod_date="2021-05-29T10:24:16Z"
   last_mod_user="#id77" license_list="" lsd="2021-05-29T10:24:16Z"
object_application="Teamcenter"
   object_desc="" object_name="one-View" object_properties="0" object_type="BOMView"
owning_group="#id75"
   owning_organization="" owning_project="" owning_user="#id77" parent_item="#id88"
parent_uid="#id88"
   pid="594" process_stage_list="" project_list="" release_status_list="" revision_limit="1"
   revision_number="0" timestamp="Q5Nx6UvxI45M1A" view_type="#id78" wso_thread="">
   <GSIdentity elemId="id90" label="LegacyIDid90one" />
  </PSBOMView>
- <PSOccurrenceThread clone_stable_occ_uid="AAAAAAAAAAAAAA" elemId="id17" island_id="3"
   lsd="2021-05-29T10:24:26Z" object_properties="0" parent_uid="#id89" pid="256"
timestamp="gBDx6UvxI45M1A">
   <GSIdentity elemId="id91" label="LegacyIDid91one" />
  </PSOccurrenceThread>
- <PSBOMViewRevision acl_bits="0" active_seq="1" archive_date="" archive_info=""
backup_date=""
   bom_view="#id90" creation_date="2021-05-29T10:24:17Z" date_released="" ead_paragraph=""
elemId="id18"
   gov_classification="" ip_classification="" is_precise="0" island_id="3"
   last_mod_date="2021-05-29T10:24:26Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:24:26Z"
   object_application="Teamcenter" object_desc="" object_name="one/A-View"
object_properties="0"
   object_type="BOMView Revision" owning_group="#id75" owning_organization=""
owning_project=""
   owning_user="#id77" parent_uid="#id87" pid="595" process_stage_list="" project_list=""
   release_status_list="" revision_limit="1" revision_number="0"
struct_last_mod_date="2021-05-29T10:24:26Z"
   timestamp="gBCx6UvxI45M1A" wso_thread="">
   <GSIdentity elemId="id92" label="LegacyIDid92one" />
  </PSBOMViewRevision>
- <Anchor acl_bits="0" archive_date="" archive_info="" backup_date=""
creation_date="2021-05-29T10:23:35Z"
   elemId="id19" immune_objects="" island_id="3" keep_limit="3"
last_mod_date="2021-05-29T10:23:35Z"
   last_mod_user="#id77" lsd="2021-05-29T10:23:35Z" managed_objects="" object_properties="0"
   owning_group="#id75" owning_user="#id77" parent_uid="#id87" pid="520"
timestamp="QtIx6UvxI45M1A">
   <GSIdentity elemId="id93" label="LegacyIDid93one" />
  </Anchor>
```

```
- <ImanRelation elemId="id20" island_id="3" lsd="2021-05-29T10:23:35Z" object_properties="0"
parent_uid="#id88"
   pid="443" primary_object="#id88" relation_type="#id76" secondary_object="#id96"
timestamp="QtCx6UvxI45M1A"
   user_data="">
   <GSIdentity elemId="id94" label="LegacyIDid94one" />
  </ImanRelation>
- <ImanRelation elemId="id21" island_id="3" lsd="2021-05-29T10:23:35Z" object_properties="0"
parent_uid="#id87"
   pid="443" primary_object="#id87" relation_type="#id76" secondary_object="#id97"
timestamp="QtPx6UvxI45M1A"
   user_data="">
   <GSIdentity elemId="id95" label="LegacyIDid95one" />
  </ImanRelation>
- <Form acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:35Z" data_file="" date_released="" ead_paragraph=""
elemId="id22"
   form_file="n/a" gov_classification="" ip_classification="" island_id="3"
   last_mod_date="2021-05-29T10:23:35Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:23:35Z"
   object_application="Teamcenter" object_desc="" object_name="one" object_properties="0"
   object_type="Item Master" owning_group="#id75" owning_organization="" owning_project=""
owning_user="#id77"
   parent_uid="#id94" pid="555" process_stage_list="" project_list="" release_status_list=""
   revision_limit="1" revision_number="0" timestamp="QpPx6UvxI45M1A" wso_thread="">
   <GSIdentity elemId="id96" label="LegacyIDid96one" />
  </Form>
- <Form acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:35Z" data_file="" date_released="" ead_paragraph=""
elemId="id23"
   form_file="n/a" gov_classification="" ip_classification="" island_id="3"
   last_mod_date="2021-05-29T10:23:35Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:23:35Z"
   object_application="Teamcenter" object_desc="" object_name="one/A" object_properties="0"
   object_type="ItemRevision Master" owning_group="#id75" owning_organization=""
owning_project=""
   owning_user="#id77" parent_uid="#id95" pid="555" process_stage_list="" project_list=""
   release_status_list="" revision_limit="1" revision_number="0" timestamp="QtMx6UvxI45M1A"
wso_thread="">
   <GSIdentity elemId="id97" label="LegacyIDid97one" />
  </Form>
- <Header author="tcdba" date="2021-05-29" elemId="id24" originatingSite="-2132200621"
targetSite=""
   time="15:58:52">
- <TransferFormula elemId="id25">
- <OptionSet elemId="id26" name="SiteConsolidationDefault">
  <Option elemId="id27" name="internalClosureRule" value="True" />
  <Option elemId="id28" name="opt_entire_bom" value="True" />
  <Option elemId="id29" name="opt_entire_mse" value="False" />
  <Option elemId="id30" name="opt_ixr_islanchor" value="False" />
  <Option elemId="id31" name="opt_mechatro" value="False" />
  <Option elemId="id32" name="opt_res_audit" value="True" />
  <Option elemId="id33" name="opt_res_checkout" value="False" />
  <Option elemId="id34" name="opt_varexp_islanchor" value="True" />
  </OptionSet>
- <SessionOptions elemId="id35">
  <Option elemId="id36" name="fastStream" value="yes" />
  </SessionOptions>
  <TransferMode elemId="id37" object_name="SiteConsolidationDefaultTM" />
  <Reason elemId="id38" />
```

```
</TransferFormula>
</Header>
</TCXML>
```

Validating TC XML before import

Pre-import validation utility

The TC XML file generated by the custom mapping solution needs to be validated to ensure that it is constructed properly for consumption by fast import. The **tcxml_validate** utility serves this purpose by performing the following validations:

1. Checks for well-formed XML.
2. Validates XML against XML schema.
3. Checks for well-formed and valid UIDs.
4. Validates typed and untyped references.
5. Checks for missing attributes,
6. Checks for required helper objects configured by closure rules.
7. Checks for conformance to GRM rules.

Ensure access to the validation schema

The **tcxml_validate** utility uses the **LLTCMXML.xsd** schema to validate the input files. This file must be in the *TC_DATA* directory. To generate this file and enable schema validation:

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

1. Use the **bmide_generatetcplmxmlschema** utility to generate the file for your Teamcenter site, for example:

   ```
   bmide_generatetcplmxmlschema -u=tc-admin-user -p=password -g=group
   -schema_type=llschema
   ```

   The utility may take a few minutes to finish and displays some information in the command shell as it progresses.

2. Verify the **LLTCXML.xsd** file is in the *TC_DATA* directory.

   If the file is not in that directory, verify you have the Business Modeler IDE application properly installed and rerun the utility.

3. Set the Teamcenter **TIE_validate_xml_against_schema** preference to **true**.

Required helper objects

You use closure rules define the objects that are mandatory to form a valid data model at your Teamcenter site. The standard (OOTB) closure rules are marked by the required (**R**) predicate. Any new closure rule clauses you add for custom data model objects must be marked with the required predicate by suffixing the clause with **:R**. For example, the following explains the standard closure rule clauses for **ItemRevision** and **Variant** class objects

Every **ItemRevision** object must have an **Item** object represented by the following clause with the **:R** predicate:

```
CLASS.ItemRevision:CLASS.Item:ATTRIBUTE.items_tag:PROCESS+TRAVERSE:  :I+R
```

Every **Variant** object must have a **parent_Item** object represented by:

```
CLASS.Item:CLASS.Variant:REFBY.parent_item:PROCESS+TRAVERSE: :R
```

Validate your TC XML test files

Use **tcxml_validate** utility to validate the TC XML file generated by the custom mapping solution. When you first run the utility, use input files that have small data samples to verify the correctness of the custom mapping solution.

The **tcxml_validate** utility is intended as a tool used to perfect your custom mapping solution. It is not intended to be used on all TC XML files before they are imported.

The *input-file-name*_**validator.log** contains a report of any errors found in the input TC XML file. Your mapping solution must be fixed to clear these errors. After correcting the custom mapping, rerun the **tcxml_validate** utility to validate the fixes. Use this process recursively on a small, but representative, set of data. Continue the process until the mapping solution is perfected. Once all aspects of the mapping are correct, the TC XML files can be generated in bulk by the mapping solution.

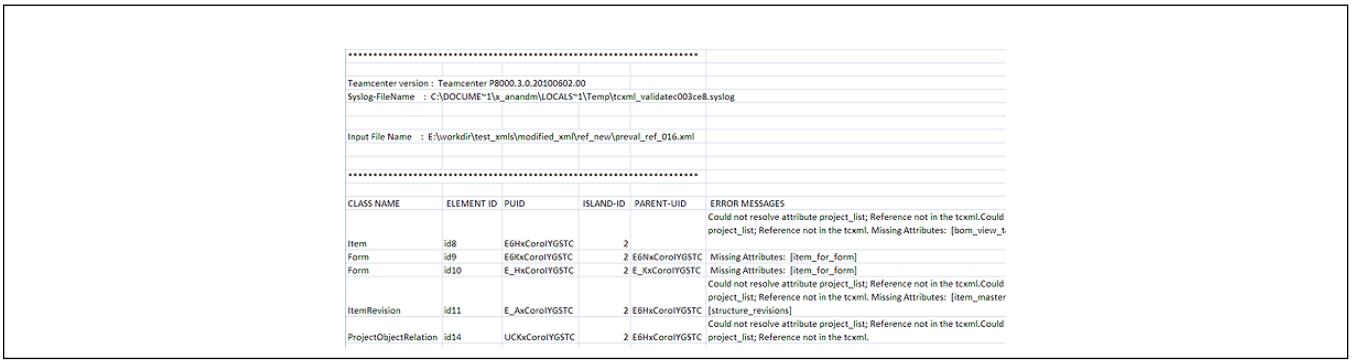
> Note:
>
> You cannot use the **tcxml_validate** utility on very large **TC XML** files. It is limited to files containing approximately 2500 elements.

For a complete description of the **tcxml_validate** utility, including syntax and examples, see the *Teamcenter Utilities*.

Pre-import validation logs

The **tcxml_validate** utility creates a log file containing the results of validation at the location of the input TC XML file. The file naming convention is *input-xml-file*_**validator.log**.

The log file contains the errors found for an XML element along with the details such as **Object UID**, **Class**, **Island-id**, and parent object. The values for an element are separated by pipe symbol (**|**). The log file can be imported into an Excel worksheet. Set the delimiter to a bar (**|**) or pipe character when importing it into Excel for better readability. The following figure shows a sample log after it is imported into Excel:



Teamcenter version : Teamcenter P8000.3.0.20100602.00
Syslog-FileName : C:\DOCUME~1\x_anandm\LOCALS~1\Temp\tcxml_validatec003ce8.syslog

Input File Name : E:\workdir\test_xmls\modified_xml\ref_new\preval_ref_016.xml

CLASS NAME	ELEMENT ID	PUID	ISLAND-ID	PARENT-UID	ERROR MESSAGES
Item	id8	E6HxCorolYGSTC	2		Could not resolve attribute project_list; Reference not in the tcxml.Could project_list; Reference not in the tcxml. Missing Attributes: [bom_view_t
Form	id9	E6KxCorolYGSTC	2	E6NxCorolYGSTC	Missing Attributes: [item_for_form]
Form	id10	E_HxCorolYGSTC	2	E_KxCorolYGSTC	Missing Attributes: [item_for_form]
ItemRevision	id11	E_AxCorolYGSTC	2	E6HxCorolYGSTC	Could not resolve attribute project_list; Reference not in the tcxml.Could project_list; Reference not in the tcxml. Missing Attributes: [item_master [structure_revisions]
ProjectObjectRelation	id14	UCKxCorolYGSTC	2	E6HxCorolYGSTC	Could not resolve attribute project_list; Reference not in the tcxml.Could project_list; Reference not in the tcxml.

Validation log file in Excel

Caution:

On Windows systems, when schema validation is used and the TC XML elements have some mandatory attributes missing, an exact error message is not included in the log file. However, the log file contains the line number and column number of the missing attributes in the file, for example:

```
TIEFastImportSAXHandler Error: [] encountered in xml at line
[17] and column [898]. May have encountered some invalid
characters or missing some required attributes - but anyway
attempting to continue processing...
```

To determine the missing mandatory attributes, check the element listed in the indicated line and see the appropriate object table in *Teamcenter core data dictionary*.

Using the bulk data loader utility

Bulk loader utility

The **tcxml_import** utility in bulk loader mode imports legacy data in a TC XML file into Teamcenter. The utility can be run only by a system administrator (Teamcenter user in the **dba** group) and requires a license key. (Open a support case on Support Center to get this license key.) The objects are imported

as local objects by providing the **-bulk_load** argument. In case of any failures during import, a file with failed islands is created in the same location as that of the TC XML input file.

For bulk loading using the **tcxml_import** utility:

> Note:
>
> The syntax defined for this utility in *Teamcenter Utilities* contain arguments not listed here. Those arguments are not required for bulk loading of legacy data. They are ignored when the **-bulk_load** argument is used.

Syntax

tcxml_import [**-u=***user-id* {**-p=***password* **| -pf=***password-file*} **-g=***group*]
-file=*xml-file-name* **-site=***site-name*
-bulk_load
[**-h**]

Arguments

-u

Specifies the user ID.

This is a user with Teamcenter administration privileges. If this argument is used without a value, the operating system user name is used.

> Note:
>
> If Security Services single sign-on (SSO) is enabled for your server, the **-u** and **-p** arguments are authenticated externally through SSO rather than being authenticated against the Teamcenter database. If you do not supply these arguments, the utility attempts to join an existing SSO session. If no session is found, you are prompted to enter a user ID and password.

-p

Specifies the password.

If used without a value, the system assumes a null value. This argument is mutually exclusive with the **-pf** argument.

If this argument is not used, the system assumes the *user-id* value to be the password.

-pf

Specifies the password file. This file can be created manually, or using Teamcenter Environment Manager.

- If created manually, the file must be a single-line ASCII file containing the password in clear text.
- If created through Teamcenter Environment Manager, the wizard prompts you for a password and creates the password file during installation.

If used without a value, the system assumes a null value. If this argument is not used, the system assumes the *user-id* value to be the password.

This argument is mutually exclusive with the **-p** argument.

-g

Specifies the group associated with the user.

If used without a value, the user's default group is assumed.

-file

Specifies the input TC XML file containing the objects to be imported into Teamcenter.

-site

Specifies the exporting (owning) site from which input TC XML data is generated.

-bulk_load

Performs a fast import using POM level APIs of a file containing legacy data objects. The file must conform to the TC XML low-level data format. You must specify the **-file** and **-site** arguments. If you specify any other optional arguments they are ignored. See **restrictions**.

-h

Displays help for this utility.

Restrictions

1. Using POM level APIs ensures the overall integrity of the data (referential integrity, type safety, and so forth). However, the **tcxml_import** utility does not verify adherence to business rules. You must identify these types of errors and fix them during the **pre-import process**.

2. When importing a **PSOccurrence** object, the import file must contain the following attributes:

   - **xform**
   - **notes_ref**
   - **child_bv**
   - **variant_condition**
   - **alternate_etc_ref**

3. When importing a workspace object, the following properties must be present in the import file:

   - **uom_tag**
   - **configuration_object_tag**

Examples

- To import legacy objects from the Item32.xml file into the Teamcenter site where you run the utility:

  ```
  tcxml_import –u=tc-admin-user -p=password -file=D:\Temp\Item32.xml
  -bulk_load -site=-2140766078
  ```

  The following files are created at the same location as the input XML file:

  - **Item32_importer.log**

    Contains the import process details.

  - **Item32_pretraverse.log**

    Contains the results of pre-traversal performed at the importing site.

  - **Item32_import_results.txt**

    Contains the import status (success or failure of island).

  - **Item32_repeat.xml**

    > Note:
    >
    > This file is created only when failures occur during the import process.

    Contains data for the failed islands, along with all organizational elements and headers. Additionally it has a comment tag to display the failure message.

- To upload dataset files from a local directory into the importing site when the **Item32.xml** file contains **ImanFile** elements:

  ```
  tcxml_import –u=tc-admin-user -p=password -file=D:\Temp\Item32.xml
  -bulk_load
  ```

Sample bulk load repeat file

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <TCXML xmlns="http://www.tcxml.org/Schemas/TCXMLSchema">
- <!--
 Summary of island failure errors:  Failed with Error The instance cannot be saved because
it contains at least
 one attribute that violates a unique attribute rule.(515106) for Island: 3(Primary
Object:gYAx6YHEaQAhNA)
  -->
- <Group elemId="id1" island_id="0" name="dba">
  <GSIdentity elemId="id75" label="LegacyIDid75one" />
```

```
  </Group>
- <ImanType elemId="id2" island_id="0" type_class="ImanRelation"
type_name="IMAN_master_form">
  <GSIdentity elemId="id76" label="LegacyIDid76one" />
  </ImanType>
- <User elemId="id3" island_id="0" user_id="tcdba">
  <GSIdentity elemId="id77" label="LegacyIDid77one" />
  </User>
- <PSViewType elemId="id4" island_id="0" name="view">
  <GSIdentity elemId="id78" label="LegacyIDid78one" />
  </PSViewType>
- <POM_imc elemId="id5" island_id="0" site_id="-1589622576">
  <GSIdentity elemId="id79" label="LegacyIDid79one" />
  </POM_imc>
- <ItemRevision acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:35Z" date_released="" declared_options=""
ead_paragraph="" elemId="id13"
   gde_bvr_list="" gov_classification="" has_variant_module="" ip_classification=""
island_id="3"
   item_revision_id="A" items_tag="#id88" last_mod_date="2021-05-29T10:24:26Z"
last_mod_user="#id77"
   license_list="" lsd="2021-05-29T10:24:26Z" object_application="Teamcenter" object_desc=""
object_name="one"
   object_properties="0" object_type="ItemRevision" owning_group="#id75"
owning_organization=""
   owning_project="" owning_user="#id77" parent_uid="#id88" pid="758" process_stage_list=""
project_list=""
   release_status_list="" revision_limit="1" revision_number="0" sequence_anchor="#id93"
sequence_id="1"
   sequence_limit="3" structure_revisions="#id92" timestamp="gBLx6UvxI45M1A" used_options=""
   variant_expression_block="" wso_thread="">
  <GSIdentity elemId="id87" label="LegacyIDid87one" />
  </ItemRevision>
- <Item acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
bom_view_tags="#id90"
   configuration_object_tag="" creation_date="2021-05-29T10:23:35Z" date_released=""
ead_paragraph=""
   elemId="id14" global_alt_list="" gov_classification="" has_variant_module=""
ip_classification=""
   is_configuration_item="" is_vi="" island_id="3" item_id="one"
last_mod_date="2021-05-29T10:24:26Z"
   last_mod_user="#id77" license_list="" lsd="2021-05-29T10:24:26Z"
object_application="Teamcenter"
   object_desc="" object_name="one" object_properties="0" object_type="Item"
owning_group="#id75"
   owning_organization="" owning_project="" owning_user="#id77" parent_uid="" pid="564"
preferred_global_alt=""
   process_stage_list="" project_list="" release_status_list="" revision_limit="1"
revision_number="0"
   timestamp="gBMx6UvxI45M1A" uom_tag="" wso_thread="">
  <GSIdentity elemId="id88" label="LegacyIDid88one" />
  </Item>
- <PSOccurrence alternate_etc_ref="" child_bv="" child_item="gUJx6YHEaQAhNA"
effectivities="" elemId="id15"
   island_id="3" lsd="2021-05-29T10:24:26Z" notes_ref="" object_properties="0" occ_flags="0"
occ_thread="#id91"
   occ_type="" occurrence_type="0" order_no="10" parent_bvr="#id92" parent_uid="#id92"
pid="783" pred_list=""
   qty_value="-1" seq_no="10" timestamp="gBEx6UvxI45M1A" uom_tag="" used_options=""
variant_condition=""
```

```
   xform="">
  <GSIdentity elemId="id89" label="LegacyIDid89one" />
  </PSOccurrence>
- <PSBOMView acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:24:16Z" date_released="" ead_paragraph="" elemId="id16"
gov_classification=""
   ip_classification="" island_id="3" last_mod_date="2021-05-29T10:24:16Z"
last_mod_user="#id77"
   license_list="" lsd="2021-05-29T10:24:16Z" object_application="Teamcenter" object_desc=""
   object_name="one-View" object_properties="0" object_type="BOMView" owning_group="#id75"
   owning_organization="" owning_project="" owning_user="#id77" parent_item="#id88"
parent_uid="#id88"
   pid="594" process_stage_list="" project_list="" release_status_list="" revision_limit="1"
   revision_number="0" timestamp="Q5Nx6UvxI45M1A" view_type="#id78" wso_thread="">
  <GSIdentity elemId="id90" label="LegacyIDid90one" />
  </PSBOMView>
- <PSOccurrenceThread clone_stable_occ_uid="AAAAAAAAAAAAAA" elemId="id17" island_id="3"
   lsd="2021-05-29T10:24:26Z" object_properties="0" parent_uid="#id89" pid="256"
timestamp="gBDx6UvxI45M1A">
  <GSIdentity elemId="id91" label="LegacyIDid91one" />
  </PSOccurrenceThread>
- <PSBOMViewRevision acl_bits="0" active_seq="1" archive_date="" archive_info=""
backup_date=""
   bom_view="#id90" creation_date="2021-05-29T10:24:17Z" date_released="" ead_paragraph=""
elemId="id18"
   gov_classification="" ip_classification="" is_precise="0" island_id="3"
last_mod_date="2021-05-29T10:24:26Z"
   last_mod_user="#id77" license_list="" lsd="2021-05-29T10:24:26Z"
object_application="Teamcenter"
   object_desc="" object_name="one/A-View" object_properties="0" object_type="BOMView
Revision"
   owning_group="#id75" owning_organization="" owning_project="" owning_user="#id77"
parent_uid="#id87"
   pid="595" process_stage_list="" project_list="" release_status_list="" revision_limit="1"
   revision_number="0" struct_last_mod_date="2021-05-29T10:24:26Z"
timestamp="gBCx6UvxI45M1A" wso_thread="">
  <GSIdentity elemId="id92" label="LegacyIDid92one" />
  </PSBOMViewRevision>
- <Anchor acl_bits="0" archive_date="" archive_info="" backup_date=""
creation_date="2021-05-29T10:23:35Z"
   elemId="id19" immune_objects="" island_id="3" keep_limit="3"
last_mod_date="2021-05-29T10:23:35Z"
   last_mod_user="#id77" lsd="2021-05-29T10:23:35Z" managed_objects="" object_properties="0"
   owning_group="#id75" owning_user="#id77" parent_uid="#id87" pid="520"
timestamp="QtIx6UvxI45M1A">
  <GSIdentity elemId="id93" label="LegacyIDid93one" />
  </Anchor>
- <ImanRelation elemId="id20" island_id="3" lsd="2021-05-29T10:23:35Z" object_properties="0"
   parent_uid="#id88" pid="443" primary_object="#id88" relation_type="#id76"
secondary_object="#id96"
   timestamp="QtCx6UvxI45M1A" user_data="">
  <GSIdentity elemId="id94" label="LegacyIDid94one" />
  </ImanRelation>
- <ImanRelation elemId="id21" island_id="3" lsd="2021-05-29T10:23:35Z" object_properties="0"
   parent_uid="#id87" pid="443" primary_object="#id87" relation_type="#id76"
secondary_object="#id97"
   timestamp="QtPx6UvxI45M1A" user_data="">
  <GSIdentity elemId="id95" label="LegacyIDid95one" />
  </ImanRelation>
- <Form acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
```

```
   creation_date="2021-05-29T10:23:35Z" data_file="" date_released="" ead_paragraph=""
elemId="id22"
   form_file="n/a" gov_classification="" ip_classification="" island_id="3"
   last_mod_date="2021-05-29T10:23:35Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:23:35Z"
   object_application="Teamcenter" object_desc="" object_name="one" object_properties="0"
   object_type="Item Master" owning_group="#id75" owning_organization="" owning_project=""
   owning_user="#id77" parent_uid="#id94" pid="555" process_stage_list="" project_list=""
   release_status_list="" revision_limit="1" revision_number="0" timestamp="QpPx6UvxI45M1A"
wso_thread="">
  <GSIdentity elemId="id96" label="LegacyIDid96one" />
  </Form>
- <Form acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date=""
   creation_date="2021-05-29T10:23:35Z" data_file="" date_released="" ead_paragraph=""
elemId="id23"
   form_file="n/a" gov_classification="" ip_classification="" island_id="3"
   last_mod_date="2021-05-29T10:23:35Z" last_mod_user="#id77" license_list=""
lsd="2021-05-29T10:23:35Z"
   object_application="Teamcenter" object_desc="" object_name="one/A" object_properties="0"
   object_type="ItemRevision Master" owning_group="#id75" owning_organization=""
owning_project=""
   owning_user="#id77" parent_uid="#id95" pid="555" process_stage_list="" project_list=""
   release_status_list="" revision_limit="1" revision_number="0" timestamp="QtMx6UvxI45M1A"
wso_thread="">
  <GSIdentity elemId="id97" label="LegacyIDid97one" />
  </Form>
- <Header author="tcdba" date="2021-05-29" elemId="id24" originatingSite="-2132200621"
targetSite=""
   time="15:58:52">
- <TransferFormula elemId="id25">
- <OptionSet elemId="id26" name="SiteConsolidationDefault">
  <Option elemId="id27" name="internalClosureRule" value="True" />
  <Option elemId="id28" name="opt_entire_bom" value="True" />
  <Option elemId="id29" name="opt_entire_mse" value="False" />
  <Option elemId="id30" name="opt_ixr_islanchor" value="False" />
  <Option elemId="id31" name="opt_mechatro" value="False" />
  <Option elemId="id32" name="opt_res_audit" value="True" />
  <Option elemId="id33" name="opt_res_checkout" value="False" />
  <Option elemId="id34" name="opt_varexp_islanchor" value="True" />
  </OptionSet>
- <SessionOptions elemId="id35">
  <Option elemId="id36" name="fastStream" value="yes" />
  </SessionOptions>
  <TransferMode elemId="id37" object_name="SiteConsolidationDefaultTM" />
  <Reason elemId="id38" />
 </TransferFormula>
 </Header>
 </TCXML>
```

Frequently asked questions about bulk data loading

Consideration	Description
Why is there a different referencing scheme used within the TC XML format?	To expand on this question with examples, for **owning_user** the element id is used, for the referencing **items_tag**, the **uid** of the parent id is used. Also the **parent_uid** attribute is mentioned twice as **items_tag** and **parent_uid**. To be able to separate the users/objects/

Consideration	Description
	relations in the loads it would be best to reference everything by the Teamcenter UID. `<ItemRevision acl_bits="0" active_seq="1" archive_date="" archive_info="" backup_date="" creation_date="2010-06-29T15:43:21Z" date_released="" declared_options="" ead_paragraph="" elemId="id17" gde_bvr_list="" gov_classification="" has_variant_module="" ip_classification="" island_id="4" item_revision_id="A" items_tag="AVDxiag7I45M1A" last_mod_date="2010-06-29T15:46:31Z" last_mod_user="#id5" license_list="" lsd="2010-06-29T15:46:31Z" object_application="Teamcenter" object_desc="" object_name="child1" object_properties="0" object_type="ItemRevision" owning_group="#id1" owning_organization="" owning_project="" owning_user="#id5" parent_uid="AVDxiag7I45M1A" pid="758" process_stage_list="" project_list="" puid="AVMxiag7I45M1A" release_status_list="" revision_limit="1" revision_number="0" sequence_anchor="AVOxiag7I45M1A" sequence_id="1" sequence_limit="3" structure_revisions="QFJxiag7I45M1A" timestamp="QNMxiag7I45M1A" used_options="" variant_expression_block="" wso_thread=""/>` Organization and administrative objects, such as **User** and **Group**, cannot be referenced by a **UID**. They have to be looked up based on their candidate key. So there is an XML element for the organization object with just the candidate key and element ID and all the references use the **elemId** attribute to point to it.
How are the island IDs handled?	The import process collects all the objects that have the same **island-id** attribute and saves them at once.
Why not write header information to a separate configuration file?	The import process relies on originating site information and the session options in the **Header** element. This information is read in the first pass of the XML parsing and used in the second pass when the actual import happens.
Which attributes in the TC XML are mandatory? Can empty attributes can be left out?	The fast import sets null values for missing attributes that do not have an initial value. However, there is no straightforward way to arrive at a set of attributes that are required to be populated for a given class. The **tcxml_validate** utility reports if the required typed and untyped references are missing from the TC XML. In addition, the list of mandatory attributes for core Teamcenter classes documented in *Teamcenter core data dictionary*. For other classes of interest, similar exercise must be done as part of the customization work.
How is the time stamp generated?	The time stamp shows when the object was last modified. The **timestamp** attribute takes the form of a UID – not a date or time as

Consideration	Description
	might be expected. Because a UID encodes a representation of time, a UID is sufficient for this purpose. This attribute can be omitted from the TC XML. Fast import internally generates and sets the time stamp for an object.
What are the restrictions for the external and site ID values used with **TIE_get_hashed_uid**?	When generating the UID with **TIE_get_hashed_uid**, an external ID coming from the legacy system and a site ID identifying the legacy system are required. These ID values have no character set or string length restrictions.
Is the hashed UID the same as the external ID used for creating the UID internally with the bulk loader process? Or what is the format of the GSID needed?	The GSID contains **label**, **sublabel**, and **split_token** attributes to allow for many-to-many mappings. If the external ID is composed such that it is unique across the legacy data model it can be used to populate the **label** attribute.
Must the **elemId** and **island_id** attributes be unique in a file or unique over all data loaded?	The **elemId** and **island_id** attributes are unique per file.
What causes errors during job scheduling and execution?	Process execution problems may occur due to available memory or maximum time-to-live (ttl) expiration when running in the Apache ODE in memory. This issue is usually encountered when you are running the ODE on a JBoss application server and the ODE uses an Oracle database. To run an ODE process in memory, uncomment or add the **<in-memory>true</in-memory>** element to the *ode-working-dir***/processes/data-transfer/deploy.xml** file. For more information about ODE processes, see **https://ode.apache.org**. If your processes may exist for longer than 10 minutes, configure the in-memory ttl to longer than 10 minutes using the **ode-axis2.mex.inmem.ttl** property. This properties value is set in milliseconds in the *ode-working-dir***/conf/ode-axis2.properties** file. For example, the following set the ttl to 30 minutes: `ode-axis2.mex.inmem.ttl=1800000`

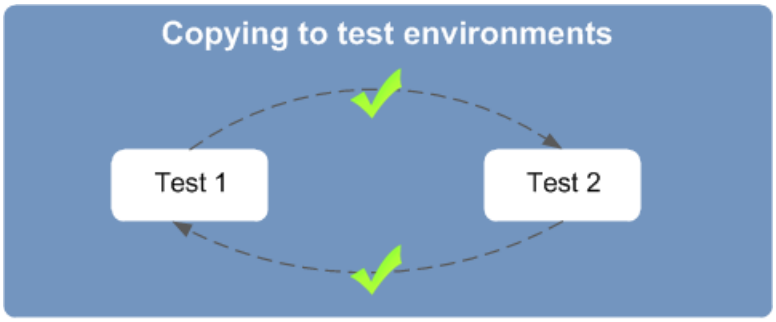
4. Copying product data to test environments

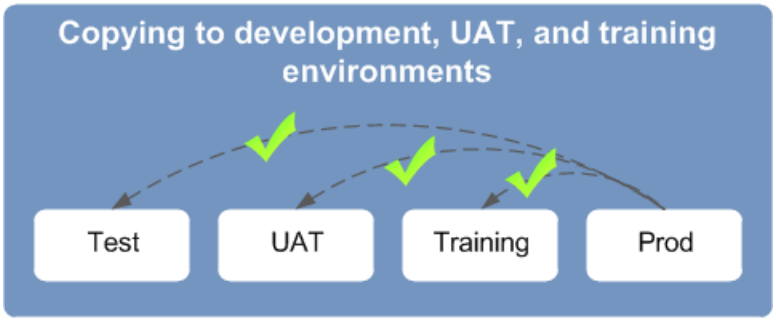
Extracting and loading bulk production data

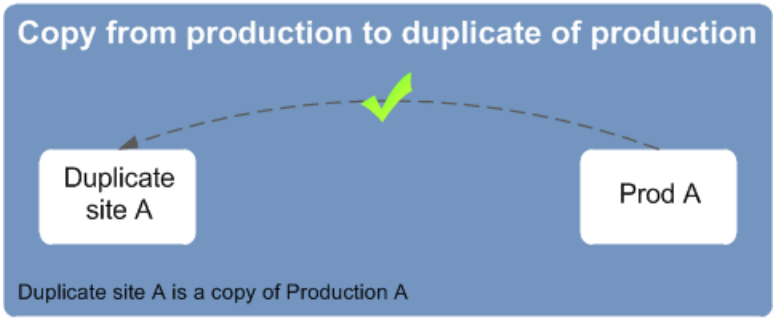
Teamcenter provides tools to extract large amounts of data from one Teamcenter environment to copy to another. You would typically use these tools and data in the following situations:

- To seed a developer environment with actual production data for testing of customizations.
- To test point releases or to train users on such releases.
- To practice upgrading a production environment using actual production data prior to rolling out the upgrade to the production sites.
- To troubleshoot an issue using actual product data in an isolated environment.

The following site imports using low-level bulk loading tools are supported.

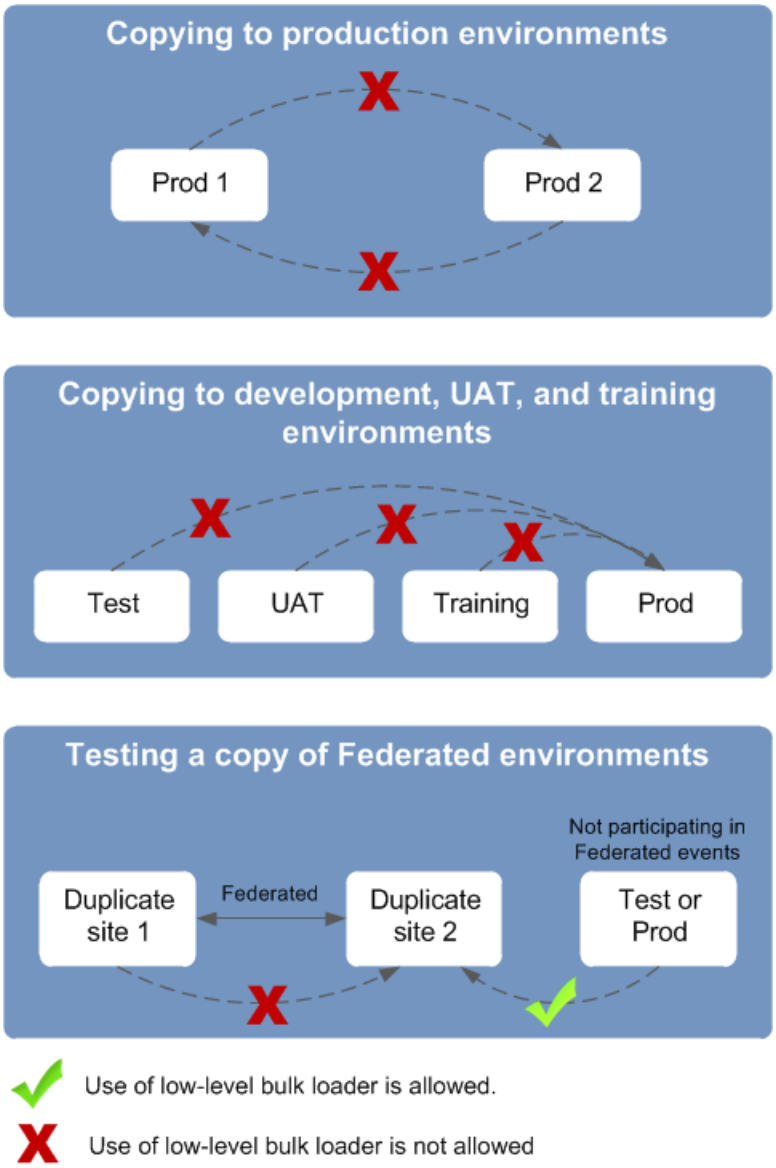






Use of low-level bulk loader is allowed.

Use of low-level bulk loader is not allowed.



Product data bulk extract and bulk load functions

You can move selected product data from a production environment to test or training environment using the bulk load features of Teamcenter. The bulk load feature is accessible using the **tcxml_export** and **tcxml_import** utilities and uses the low-level TC XML format to quickly extract and load the selected data. Additionally, you can access this feature using the **Bulk Extract** and **Bulk Load** commands on the **Tools** menu in the rich client. Either method creates a copy of the desired data in a briefcase file that you import into the target test site.

To bulk load product data into a site, it must be designated as a test site. You can designate a site as a test site during the install process in Teamcenter Environment Manager (TEM). Teamcenter also provides the **-mark_as_test_env** argument for the **install** utility that allows you to change a site created as a

production environment to test environment. This can be used when a site was created using the default (production environment) and it is subsequently determined that it must be a test environment.

Creating a test or training environment

You can create a site containing a copy of your production data for upgrade testing, acceptance testing of customizations, or training purposes. Teamcenter Environment Manager is the recommended tool for creating a test environment that has the features you require in your production site. Use the bulk load feature to then extract a copy of the desired product data from the production site for the test site.

Copy a Teamcenter environment

An alternative to **creating a test or training environment** is to convert a production environment that matches your Teamcenter site to a test environment. Convert a production environment using the **install** utility with the **-mark_as_test_env** argument as follows.

1. Launch Teamcenter Environment Manager (TEM) from the Teamcenter installation image, not from **install** directory of your existing *TC_ROOT*.

2. Install a Teamcenter environment that matches the solutions and features in your current Teamcenter environment.

   Follow the steps for installing a Teamcenter corporate server in the appropriate server installation guide.

3. In the **Foundation** panel, accept the default **Create and populate** database option.

   Follow the instruction for the **Foundation Database** and Volume **Information** in the appropriate server installation guide.

4. In the **Foundation Settings** panel, select the **Test Environment** option.

   Complete the installation following the instructions in the appropriate server installation guide.

Extract product data from a production environment

Extract product data using the rich client

1. Launch the rich client for your production environment, select the objects you want to extract, and choose **Tools→Export→Bulk Extract**.

2. In the **Bulk Extract to Briefcase** dialog box:

   a. Type or navigate to the location where you want to save the briefcase file in the **Directory** box.

b. Select **UnconfiguredBulkExtractDefault** in the **Option Set** list.

Teamcenter provides a default name for the briefcase file in the **File Name** box that you can modify if desired. Do not change file extension.

c. If you want to change the editable **UnconfiguredBulkExtractDefault** export options, click the **Display/Set export options** button ().

> Tip:
>
> If there are no changes to the Organization objects associated with the data you are extracting, you can prevent unnecessary export of the Organization objects by clearing the **Export All Properties of Organization Object** option.

Teamcenter displays the **UnconfiguredBulkExtractDefault** dialog box.

The **Bulk Extract data to Briefcase** option is selected by default and is read-only. Only the Teamcenter site administrator can edit the read-only check boxes.

Teamcenter exports briefcase files in a compressed format to decrease the file size. Compressing the file can be time consuming. If you want to decrease the time it takes to create the briefcase file and are not concerned with the size of the file you are extracting, select the **Export Uncompressed Briefcase** check box. This option can be set as the default option in the option set you use to bulk extract data by setting the (**opt_uncompressed_bcz** option value to **TRUE**.

d. Click **OK** to close the dialog boxes.

Teamcenter displays the **Options Settings** dialog box. If the settings are correct, click **Yes** to continue.

Teamcenter creates a briefcase (.bcz) file in the directory location and displays the **Export Completed** dialog box indicating if the export is successful.

3. Ensure the briefcase file is in a location accessible to the test environment.

Extract product data using the command line

Extract product data from a production environment using the **tcxml_export** utility. Use the utility with the **-bulk_extract** argument and one or more of the following arguments that identify the root object for the extract:

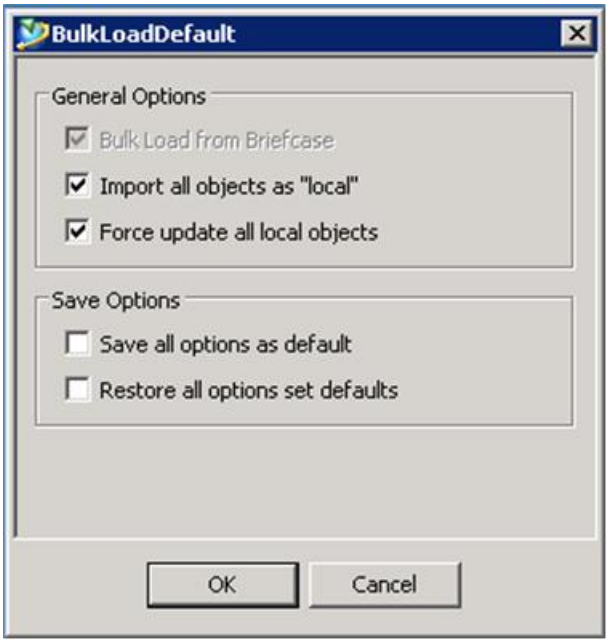
- **-input_criteria**
- **-class**
- **-folder**
- **-item**
- **-item_key**
- **-inputfile**
- **-inputuidfile**
- **-uid**

Load product data into a test environment

Load product data using the rich client

1. Launch the rich client for your test environment and choose **Tools→Import→Bulk Load**.
2. In the **Bulk Load from Briefcase** dialog box:
   a. Type or navigate to the location of the briefcase file in the **Briefcase File** box.
   b. Select **BulkLoadDefault** in the **Option Set** list.
   c. If you want to change **the editable BulkLoadDefault import options**, click the **Display/Set import options** button .

Teamcenter displays the **BulkLoadDefault** dialog box.



The **Bulk Load from Briefcase** option is selected by default and is read-only. Only the Teamcenter site administrator can edit the read-only check boxes.

3. Click **OK** to close the dialog boxes.

   Teamcenter displays the **Remote Export Options Settings** dialog box. If the settings are correct, click **Yes** to continue.

   Teamcenter imports the product data from the briefcase file into your test environment.

Load product data using the command line

Use the **tcxml_import** utility with the following arguments:

- **-bulk_load**
- **-optionset=** set to a transfer option set that has the **opt_bulk_load_bcz** options set to **TRUE**

Setting bulk loading options

Teamcenter provides the following options for bulk loading briefcase files into a test environment. If you set neither of the options, all replica objects are imported as replicas in the test environment.

- Set a test environment as the owning site of all imported objects

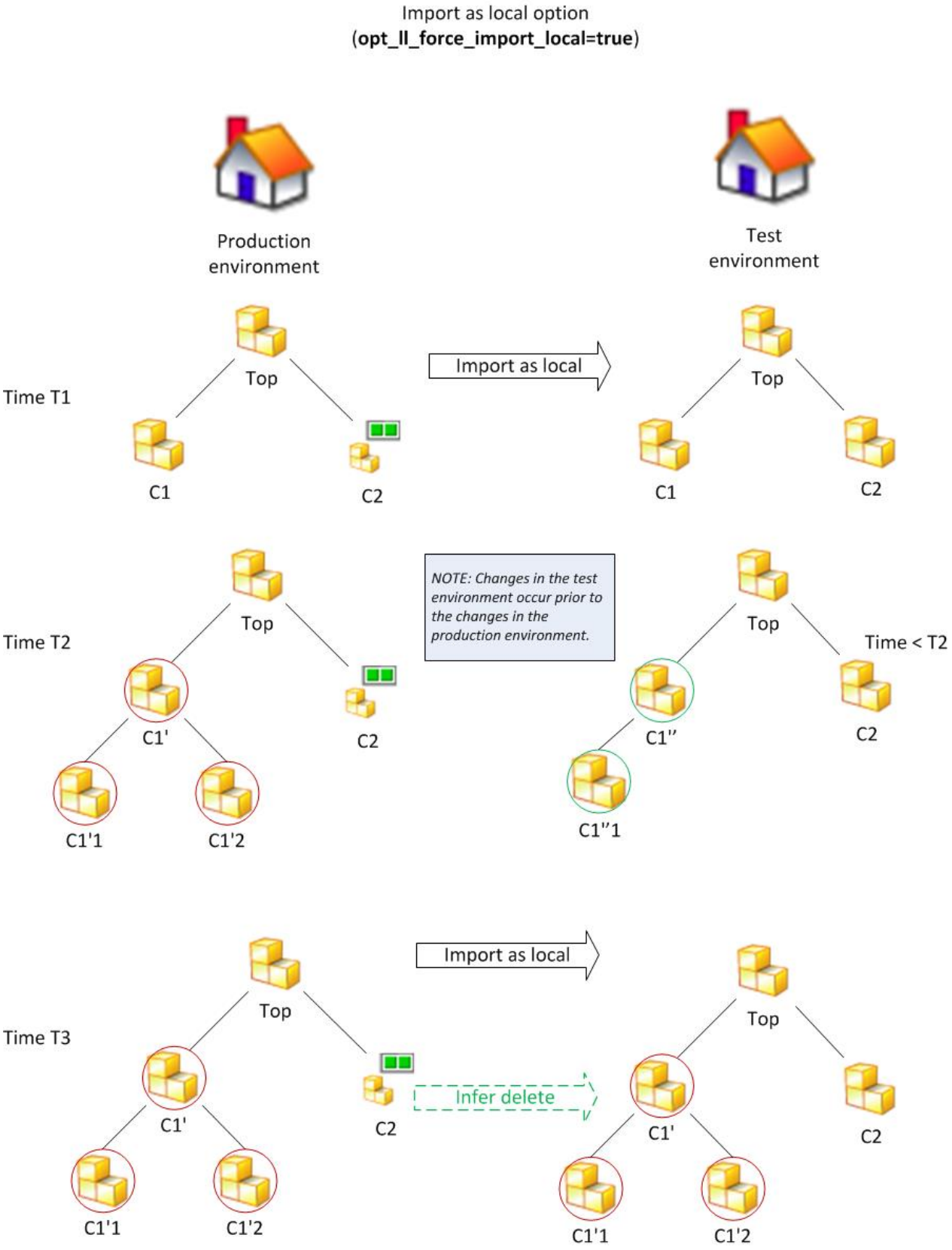
- Update all objects in a test environment with imported objects

> Caution:
>
> Take care when importing objects from different production environments that participate in a Multi-Site environment. Doing so may cause site ownership conflicts.

Set a test environment as the owning site of all imported objects

A test environment is set as the owning site for all objects in the imported briefcase file in the following circumstances:

- When selecting the **Import all objects as "local"** import option in the rich client.
- When setting on the **opt_ll_force_import_local** option to **True**.



Update all objects in a test environment with imported objects

Any objects in the test environment that also exist in the import file are replaced with the import file objects, regardless of which object was last changed (per the **lsd** attribute), in the following circumstances:

- When selecting **Force update all local objects** import option in the rich client.
- When setting on the **opt_ll_force_update_local** option to **True**.

> Warning:
>
> Using this option can cause recently updated objects in the test environment to be replaced by objects in the import file that do not contain those changes.