TEAMCENTER

Teamcenter Integration Framework

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

Teamcenter Integration Framework

Teamcenter Integration Framework overview 1-1

Installing and upgrading Teamcenter Integration Framework

Installation and update overview 2-1
Use Deployment Center to install Teamcenter Integration Framework 2-2
Use TEM to install Teamcenter Integration Framework 2-4
Use TEM to install Teamcenter Integration Framework and (optionally) Teamcenter 2-5
Upgrading and patching Teamcenter Integration Framework 2-7
- Use Deployment Center to update Teamcenter Integration Framework 2-7
- Use TEM to upgrade Teamcenter Integration Framework 2-9
- Use TEM to patch Teamcenter Integration Framework 2-9
- Resolve Teamcenter Integration Framework datastore migration conflicts 2-11
- Update custom bundles 2-16
- Retain business object definitions 2-16
- Retain JDBC site configurations as PAX-JDBC site configurations 2-17

Run Teamcenter Integration Framework 2-18
Run Teamcenter Integration Framework instances in a cluster 2-19
Run Teamcenter Integration Framework with Microsoft SQL Server and Java 11 2-23
Migrating from Global Services to Teamcenter Integration Framework 2-24
- Differences between Global Services and Teamcenter Integration Framework 2-24
- Migrate a Global Services datastore to Teamcenter Integration Framework 2-27
- Global Services to Teamcenter Integration Framework published class mapping 2-28
- Migrate a Global Services connector to Teamcenter Integration Framework 2-36
- Migrate a solution from Global Services to Teamcenter Integration Framework 2-36
- Migrate a Global Services reactor to a Teamcenter Integration Framework process 2-38
- Migrate a custom BPEL process to Groovy scripts 2-40
- Migrate Global Services email templates to Teamcenter Integration Framework 2-41

Configuring and managing Teamcenter Integration Framework operation

Configuring Teamcenter Integration Framework 3-1
- Teamcenter Integration Framework configuration overview 3-1
- Configure 3-2
- Queueing 3-11
- Activity Status 3-11
- Data Views 3-12
- Documentation 3-12
- View Default Log File 3-13

Extensions 3-13
Configuring SSO users to run Teamcenter Integration Framework configuration 3-13
Configure the integration framework email service 3-17
Troubleshoot Teamcenter Integration Framework transfer processes 3-17
Control file uploading 3-19
Stop Teamcenter Integration Framework 3-19
Teamcenter Integration Framework logging 3-19
Teamcenter Integration Framework message objects 3-19
Configuring Teamcenter Integration Framework exception message logging 3-20
Configure Teamcenter Integration Framework tracing in log files 3-21

Customizing Teamcenter Integration Framework

Teamcenter Integration Framework examples 4-1
Using Groovy scripts to customize Teamcenter Integration Framework 4-1
Groovy scripting environment 4-1
Teamcenter Integration Framework processes 4-2
Creating a Groovy process 4-3
Requests and responses 4-4
Creating a custom connector extension using Groovy 4-8
Using message-oriented middleware solutions and scripting 4-9
Teamcenter Integration Framework and message-oriented middleware 4-9
Create listeners and queues with scripts 4-10
Create Teamcenter Integration Framework queues 4-11
Monitor Teamcenter Integration Framework queues 4-12
Manage Teamcenter Integration Framework jobs 4-13
Manage Teamcenter Integration Framework queues 4-14
Teamcenter Integration Framework properties 4-15
Track activity status 4-16
Teamcenter Integration Framework solution support 4-17
Use JAXRS scripts in Teamcenter Integration Framework 4-18
Teamcenter Integration Framework connectors 4-19
Understanding Teamcenter Integration Framework connectors 4-19
Configuring and managing connectors 4-19
Teamcenter Integration Framework connector configuration 4-20
Create and manage configuration files 4-21
Add a custom connector to Teamcenter Integration Framework 4-22
Extend a connector in Teamcenter Integration Framework 4-24
Configuring wildcard characters 4-25
Data object repository connector 4-27
Teamcenter SOA connector configuration elements 4-29
Teamcenter Enterprise connector configuration elements 4-36
Teamcenter Integration Framework business objects 4-39
Understanding business objects and business object definitions 4-39
Understanding user roles 4-39
Configuring and managing Teamcenter Integration Framework business objects 4-40
Create and manage business object definition files 4-40
Building a business object definition 4-42
Modify the attribute value map file 4-56

Localizing business object definitions 4-56
Business object definition configuration elements 4-59
Attribute value map configuration elements 4-123
ID map configuration elements 4-127

1. Teamcenter Integration Framework

Teamcenter Integration Framework overview

Teamcenter Integration Framework (TcIF) integrates Teamcenter with other systems, helping to automate processes which cross system boundaries. Teamcenter Integration Framework lets you leverage your existing applications and integrate new applications with Teamcenter using a high performance and scalable framework that decouples the integrations from the applications to reduce maintenance complexity.

Install Teamcenter Integration Framework

Install (or update) Teamcenter Integration Framework using Deployment Center or Teamcenter Environment Manager (TEM) as described in **Installation and update process**.

The Teamcenter integration server requires a Java development kit (JDK) during installation. For information about versions of operating systems, third-party software, Teamcenter software, and system hardware certified for your platform, see the Hardware and Software Certifications knowledge base article on Support Center.

Configure Teamcenter Integration Framework

Teamcenter Integration Framework is configured and administered using the Teamcenter Integration Framework configuration interface as described in **Configuring Teamcenter Integration Framework**.

Start Teamcenter Integration Framework

Start the Teamcenter integration server from the command line as described in **Run Teamcenter Integration Framework**.

You can also configure Teamcenter Integration Framework to run as a service on Windows as described in **Run Teamcenter Integration Framework as a service**.

2. Installing and upgrading Teamcenter Integration Framework

Installation and update overview

You can create a stand-alone Teamcenter Integration Framework or a Teamcenter Integration Framework in an existing Teamcenter environment. A stand-alone Teamcenter Integration Framework can run on a separate host from the Teamcenter server. Be aware that Teamcenter Integration Framework requires the same number of licenses as the number of the site's Teamcenter author licenses or a minimum of 20, whichever is greater.

When upgrading Teamcenter Integration Framework, ensure your other Siemens products are compatible with the updated version of Teamcenter Integration Framework. See the Teamcenter Compatibility Matrix in the Hardware and Software Certifications knowledge base article on Support Center.

Deployment Center and Teamcenter Environment Manager (TEM) can be used to install and update Teamcenter Integration Framework. The best tools and steps vary depending on your site's configuration and requirements. Use the following information to determine the best approach for your site.

Teamcenter Integration Framework installation or upgrade type	**Installation utility**		**Notes**
	TEM	**Deployment Center**	
Install as standalone (no Teamcenter)	X		
Install Teamcenter with Teamcenter Integration Framework	X		
Install on an existing Teamcenter single machine environment	X	X	Deployment Center requires a scanned Teamcenter environment.
Install on an existing Teamcenter distributed environment		X	Deployment Center requires a scanned Teamcenter environment.
Patch or upgrade from a version before 3.0	X		If you are upgrading from a release prior to Teamcenter Integration Framework 2.2, upgrade to a 3.*x* release of Teamcenter Integration Framework before upgrading to the current

Teamcenter Integration Framework installation or upgrade type	Installation utility		Notes
	TEM	Deployment Center	
			release of Teamcenter Integration Framework.
Update or patch version 3.0 (configured as standalone)	X	X	Deployment Center requires Teamcenter Integration Framework must have been previously installed using Deployment Center.
Patch 3.0 (configured as part of a cluster)	X		

Install using Deployment Center

See **Use Deployment Center to install Teamcenter Integration Framework**.

Install using TEM

Standalone (or at the same time as Teamcenter): see **Use TEM to install Teamcenter Integration Framework and (optionally) Teamcenter**.

On a machine with an existing Teamcenter installation: see **Use TEM to install Teamcenter Integration Framework**.

Update using Deployment Center

See **Use Deployment Center to update Teamcenter Integration Framework**.

Patch or upgrade using TEM

See **Use TEM to upgrade Teamcenter Integration Framework**.

Creating clusters and updating existing clusters

For instructions on creating and updating Teamcenter Integration Framework clusters, see **Run Teamcenter Integration Framework instances in a cluster**.

Use Deployment Center to install Teamcenter Integration Framework

You can use Deployment Center for a quicker and simplified Teamcenter Integration Framework installation:

- When you are installing Teamcenter Integration Framework on a machine with Teamcenter already installed.

- When installing in a Teamcenter installation in a distributed environment that can be scanned by Deployment Center.

Sites with established TEM processes may choose to **use TEM**.

For instructions on using Deployment Center, see the Deployment Center documentation.

Use the following steps to install Teamcenter Integration Framework:

1. Download both the Teamcenter Foundation and Teamcenter Integration Framework software kits from the Teamcenter download page on Support Center and place them in the Deployment Center software repository.

   Software kits supporting several configurations are available. Ensure you have the correct software kits for the operating system and version of Teamcenter and Teamcenter Integration Framework you are installing.

2. Log onto Deployment Center and click the **Software Repositories** tile to view the Teamcenter Foundation and Teamcenter Integration Framework software kits, verifying their availability in the software repository.

3. Click the **Environments** tile to display the available environments. Select an existing scanned Teamcenter environment to use to install Teamcenter Integration Framework or another existing Teamcenter environment to use.

4. Click **Deploy Software**. On the **Software** tab, add the Teamcenter Integration Framework software to the **Selected Software** list.

5. On the **Options** tab, select the options for your environment.

6. On the **Applications** tab, update your selected applications with the following available applications:

   Integration Framework Core

   The fundamental Teamcenter Integration Framework components.

   Integration Framework for Applications

   The integration with your selected Teamcenter environment.

7. On the **Components** tab, bring each component to a 100% complete status by supplying the required settings for each.

   While supplying settings, record values such as user names and passwords for later reference.

8. On the **Deploy** tab, generate the installation scripts.

9. Follow the steps in *Run the deployment scripts* in the Deployment Center documentation to run the deployment scripts for your Teamcenter Integration Framework environment.

10. After you receive the **Deployment action successfully completed** message from the console from which you ran the deployment script, Teamcenter Integration Framework is installed. Start Teamcenter Integration Framework as described in **Run Teamcenter Integration Framework**.

Use TEM to install Teamcenter Integration Framework

Use Teamcenter Environment Manager (TEM) with the following procedure to install Teamcenter Integration Framework on a machine with Teamcenter already installed.

1. Open a Teamcenter command prompt and start TEM from the *\install* directory of an existing Teamcenter environment.

2. On the **Maintenance** panel, choose **Updates Manager**.

3. On the **Apply Updates** panel, specify the location of the Teamcenter Integration Framework kit in **Update kit location**.

4. On the **Diagnostics** panel, enter a log directory and click **Run**.

5. On the **Confirmation** panel, click **Start**.

   When the copying completes, restart TEM.

6. On the **Maintenance** panel, choose **Configuration Manager**.

7. On the **Configuration Maintenance** panel, choose **Perform maintenance on an existing configuration**.

8. On the **Feature Maintenance** panel, under **Teamcenter**, choose **Add/Remove Features**.

9. On the **Feature** panel, expand **Platform Extensibility** > **Integration Framework** and choose **Integration Framework Core**.

10. Continue through the remaining panels by accepting the default values. You can click **Advanced** to choose different ports. The default values are:

    Web Server Port: 8080
    Security Port: 9001
    Web UI Port: 8040
    Active MQ Port: 61616
    Rest Services Port: 8090

If you change the default values, make note of them. You need the **Web Server Port** number when defining your sites in Teamcenter, and you need the **Web UI Port** number to access the Teamcenter Integration Framework configuration interface when you enter the address directly in a web browser.

TEM displays the **Confirmation** panel. Click **Start** to confirm the settings and start the Teamcenter Integration Framework installation. It may take several minutes to complete the installation. TEM displays a success message when the installation completes.

- On Windows systems, a shortcut icon for starting Teamcenter Integration Framework is placed on your desktop. Alternatively, you can start Teamcenter Integration Framework in a command prompt window by running the **trun** script in the **tcif\container\bin** directory.

- On Linux systems, you can start Teamcenter Integration Framework in interactive mode by running the **trun** script in the **tcif/container/bin** directory.

Use TEM to install Teamcenter Integration Framework and (optionally) Teamcenter

Use the following steps to create a stand-alone Teamcenter Integration Framework. Be aware that Teamcenter Integration Framework requires the same number of licenses as the number of the site's Teamcenter author licenses or a minimum of 20, whichever is greater.

1. If you are installing a stand-alone Teamcenter Integration Framework, or are installing Teamcenter Integration Framework at the same time as installing Teamcenter, begin your installation with the following steps:

    a. Start Teamcenter Environment Manager (TEM) from the Teamcenter software distribution image.

    b. Select the language in the **Installer Language** dialog box.

    c. Choose **Teamcenter** in the **Welcome to Teamcenter** panel.

    d. Click **Install** in the **Install / Upgrade Options** panel.

    e. On the **Media Locations** panel, optionally enter locations of any patches you want to apply during installation.

    f. Type a new identification and description in the **Configuration** panel if you do not want to use the default values.

2. In the **Features** panel, expand **Extensions→Platform Extensibility**.

    If you are connecting to Teamcenter Enterprise, select **Enterprise Integration**.

If you are connecting to Product Master Manager, select **PMM Integration**.

If you are connecting to Supplier Relationship Management, select **SRM Integration**. (**SRM Integration** requires a separate kit.)

If you are connecting to Substance Compliance, select **Substance Compliance Integration.**.

> Note:
>
> To use the Enterprise Integration, TEM must be able to access the Teamcenter Enterprise connector files (**mti.jar** and **mtiems.jar**) for the site. You must supply their location in a later step.

If you are installing in an existing Teamcenter environment, you cannot change the installation directory. Teamcenter Integration Framework is installed in the **tcif** directory under your existing *TC_ROOT* directory. Otherwise, you can enter the path to the desired directory and Teamcenter Integration Framework is installed in the **tcif** directory under the path you enter.

3. If you are installing a stand-alone Teamcenter Integration Framework, enter the path to the Java Development Kit (JDK) in the **Path** box of the **Java Development Kit** panel.

4. You can accept the default port numbers or click **Advanced** to choose different ports. The default values are:

   Web Server Port: 8080
   Security Port: 9001
   Web UI Port: 8040
   Active MQ Port: 61616
   Rest Services Port: 8090

   If you change the default values, make note of them. You need the **Web Server Port** number when defining your sites in Teamcenter, and you need the **Web UI Port** number to access the Teamcenter Integration Framework configuration interface when you enter the address directly in a web browser.

5. If you are connecting to Teamcenter Enterprise, enter the path to the Teamcenter Enterprise connector files (**mti.jar** and **mtiems.jar**) in the **Enterprise Integration Setting** panel.

   TEM displays the **Confirmation** panel. Click **Start** to confirm the settings and start the Teamcenter Integration Framework installation. It may take several minutes to complete the installation. TEM displays a success message when the installation completes.

   On Windows systems, a shortcut icon for starting Teamcenter Integration Framework is placed on your desktop. Alternatively, you can start Teamcenter Integration Framework in a command prompt window by running the **trun** script in the **tcif\container\bin** directory.

On Linux systems, you can start Teamcenter Integration Framework in interactive mode by running the **trun** script in the **tcif/container/bin** directory.

Upgrading and patching Teamcenter Integration Framework

Use Deployment Center to update Teamcenter Integration Framework

Teamcenter Integration Framework 3.0 accompanied Teamcenter 11.4. Siemens Digital Industries Software recommends using Deployment Center to update Teamcenter Integration Framework 3.0 to later versions. However, sites with established TEM processes may **choose to use TEM**.

When upgrading Teamcenter Integration Framework, your local datastore is migrated to be compatible with the current Teamcenter Integration Framework release. Once a migration successfully completes, it cannot be rerun.

Use the following steps to update Teamcenter Integration Framework 3.0 or later using Deployment Center. For instructions on using Deployment Center, see the Deployment Center documentation.

1. From the Teamcenter download page on Support Center, download the following software kits and place them in the Deployment Center software repository:

   - The version of Teamcenter Integration Framework to which you are updating.
   - The Teamcenter Foundation kit required to support the updated version of Teamcenter Integration Framework.

   Software kits supporting several configurations are available. Ensure you have the correct software kits for the your operating system and version of Teamcenter an Teamcenter Integration Framework.

2. Log onto Deployment Center and click the **Software Repositories** tile to view the Teamcenter Foundation and Teamcenter Integration Framework software kits, verifying their availability in the software repository.

3. Click the **Environments** tile to display the environments in Deployment Center. Select the environment for which you want to update Teamcenter Integration Framework.

4. Click **Deploy Software**. On the **Software** tab, add the Teamcenter Foundation software to the **Selected Software** list if it is not already listed. Then, add the Teamcenter Integration Framework software to the **Selected Software** list.

5. On the **Options** tab, review the existing options as needed.

6. On the **Applications** tab, verify the Teamcenter Integration Framework applications are marked for updating.

7. On the **Components** tab, bring each component to a 100% complete status by supplying the required settings for each.

   While supplying settings, record values such as user names and passwords for later reference.

8. On the **Deploy** tab, generate the installation scripts.

9. Follow the steps in *Run the deployment scripts* in the Deployment Center documentation to run the deployment scripts for your Teamcenter Integration Framework environment. Once you receive the **Deployment action successfully completed** message from the console from which you ran the deployment script, Teamcenter Integration Framework is updated.

   The previous Teamcenter Integration Framework is archived, remains functional, and can be started if necessary. The previous Teamcenter Integration Framework is archived at the following location:

   TcIF_ROOT`\tcif_`*<date_and_time_stamp>*

10. Evaluate the files in *TcIF_ROOT/container/etc* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated from the archived Teamcenter Integration Framework installation.

11. Evaluate the files in *TcIF_ROOT/container/deploy* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated from the archived Teamcenter Integration Framework installation.

12. **Update your custom bundles** as necessary. If custom bundles specifically use the old Teamcenter Integration Framework version, update these bundles to use the latest Teamcenter Integration Framework version.

13. **Start Teamcenter Integration Framework**. (On Linux, start Teamcenter Integration Framework from a console and not as a background process.)

    The datastore migration occurs the first time the upgraded Teamcenter Integration Framework is started. Once the migration is complete, Teamcenter Integration Framework restarts automatically. The console window shows the migration status, and migration information is captured in the Teamcenter Integration Framework log file *TcIF_ROOT/container/log/tesb.log*.

    After the migration, you may need to manually resolve migration conflicts if files in the new datastore have the same names as files in the old datastore, yet their contents differ. See **Resolve Teamcenter Integration Framework datastore migration conflicts** for guidance.

    Once conflicts are resolved, Teamcenter Integration Framework is ready for use.

Use TEM to upgrade Teamcenter Integration Framework

When upgrading Teamcenter Integration Framework, your local datastore is migrated to be compatible with the current Teamcenter Integration Framework release. Once a migration successfully completes, it cannot be rerun.

Upgrading installs Teamcenter Integration Framework in a new directory, leaving the previous Teamcenter Integration Framework installation functional. Use the following process for upgrading your instance.

1. Use Teamcenter Environment Manager (TEM) to upgrade Teamcenter Integration Framework.

2. Evaluate the files in *TcIF_ROOT/container/etc* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated.

3. Evaluate the files in *TcIF_ROOT/container/deploy* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated.

4. **Update your custom bundles** as necessary. If custom bundles specifically use the old Teamcenter Integration Framework version, update these bundles to use the latest Teamcenter Integration Framework version.

5. Start Teamcenter Integration Framework. The datastore migration occurs the first time the upgraded Teamcenter Integration Framework is started. Once the migration is complete, Teamcenter Integration Framework restarts automatically.

   The console window shows the migration status, and migration information is captured in the Teamcenter Integration Framework log file *TcIF_ROOT/container/log/tesb.log*.

   After the migration, you may need to manually resolve migration conflicts if files in the new datastore have the same names as files in the old datastore, yet their contents differ. See **Resolve Teamcenter Integration Framework datastore migration conflicts** for guidance.

   Once conflicts are resolved, Teamcenter Integration Framework is ready for use.

Use TEM to patch Teamcenter Integration Framework

When patching Teamcenter Integration Framework, your local datastore is migrated to be compatible with the current Teamcenter Integration Framework release. If your site is part of a Teamcenter Integration Framework cluster, the cluster datastore is also migrated. In addition to datastores, custom bundles, and a set of Teamcenter Integration Framework configuration files are migrated. Once a migration successfully completes, it cannot be rerun.

When an existing Teamcenter Integration Framework installation is patched, the previous Teamcenter Integration Framework installation is moved to a backup directory provided by the administrator during the Teamcenter Environment Manager (TEM) patch process. The previous Teamcenter Integration Framework installation remains functional if it is a standalone installation. If it is part of a cluster, it

continues to be functional until the shared cluster datastore is migrated to the latest version. At that point, the previous Teamcenter Integration Framework installation may no longer work with the latest files in the datastore.

After the previous Teamcenter Integration Framework installation is moved to the backup directory, the new Teamcenter Integration Framework version is installed in the original installation directory.

The steps for patching a standalone Teamcenter Integration Framework instance differ from those for patching a Teamcenter Integration Framework instance that is part of a cluster. Each process includes manually verifying that any site-specific customizations have properly migrated. Use the following general processes for patching your instance.

Patching a standalone Teamcenter Integration Framework instance

1. Use TEM to patch Teamcenter Integration Framework.

2. Evaluate the files in *TcIF_ROOT/container/etc* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated.

3. Evaluate the files in *TcIF_ROOT/container/deploy* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated.

4. **Update your custom bundles** as necessary. If custom bundles specifically use the old Teamcenter Integration Framework version, update these bundles to use the latest Teamcenter Integration Framework version.

5. Start Teamcenter Integration Framework. The datastore migration occurs the first time the patched Teamcenter Integration Framework is started. Once the migration is complete, Teamcenter Integration Framework restarts automatically.

   The console window shows the migration status, and migration information is captured in the Teamcenter Integration Framework log file *TcIF_ROOT/container/log/tesb.log*.

   After the migration, you may need to manually resolve migration conflicts if files in the new datastore have the same names as files in the old datastore, yet their contents differ. See **Resolve Teamcenter Integration Framework datastore migration conflicts** for guidance.

   Once conflicts are resolved, Teamcenter Integration Framework is ready for use.

   When patching from a release earlier than Teamcenter Integration Framework 3.0 (delivered with Teamcenter 11.4), be aware that SSO is disabled by default in the new Teamcenter Integration Framework installation and must be manually enabled if desired.

Patching a Teamcenter Integration Framework instance that is part of a cluster

1. Shut down all Teamcenter Integration Framework instances that are part of the cluster.

2. Back up the database used by the cluster.

3. One instance at a time, perform the following steps on each instance in the cluster.

   a. Use TEM to patch Teamcenter Integration Framework on the instance. (You can also use Deployment Center to update Teamcenter Integration Framework 3.0 or later.)

   b. Evaluate the files in *TcIF_ROOT/container/etc* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated.

   c. Evaluate the files in *TcIF_ROOT/container/deploy* to ensure all custom files have been migrated from the old Teamcenter Integration Framework. Manually copy any files that have not been migrated.

   d. If custom bundles specifically use the old Teamcenter Integration Framework version, update these bundles to use the latest Teamcenter Integration Framework version.

4. Once all of the instances in the cluster have been patched, start Teamcenter Integration Framework one instance at a time. Ensure that an instance has started and has successfully completed its datastore migration before starting the next instance.

   When the first instance is started, the local datastore migration occurs and is followed by the cluster datastore migration. For the subsequent instances, only their local datastores are migrated, as the cluster datastore has already been migrated. Once the datastore migration is complete on each instance, Teamcenter Integration Framework restarts automatically.

   After a migration, you may need to manually resolve migration conflicts if files in the new datastore have the same names as files in the old datastore, yet their contents differ. See **Resolve Teamcenter Integration Framework datastore migration conflicts** for guidance.

Resolve Teamcenter Integration Framework datastore migration conflicts

Errors the first time Teamcenter Integration Framework starts after an upgrade

After a migration, certain compiler errors may be reported the first time Teamcenter Integration Framework starts. The errors will be similar to:

```
unable to resolve class com.teamcenter.esb.internal.*
```

Errors such as these are expected, as updates to scripts in the datastore are updated the first time Teamcenter Integration Framework starts after an upgrade. These compilation errors occur only during the upgrade process, and compilation of scripts will be resolved by the time Teamcenter Integration Framework has completed all upgrade steps.

Errors that persist with subsequent restarts of Teamcenter Integration Framework may indicate that customized groovy scripts do not have all called libraries available in the new version of Teamcenter

Integration Framework. Do not rely on classes with "internal" in the package space, as internal libraries may change across releases of Teamcenter Integration Framework.

Migration artifacts

You must manually resolve migration conflicts when files in the new datastore have the same names as files in the old datastore, yet their contents differ. To aid in resolving these conflicts, several files related to the migration are placed in the *TcIF_ROOT/container/migrate* directory during the migration process. The directory contains a */run* directory with subdirectories uniquely named for each migration run.

During the migration, the results of the migration are displayed in a console window and logged to the following file: *TcIF_ROOT/container/log/tesb.log*.

Standalone Teamcenter Integration Framework installation:

The following files are backed up in the *TcIF_ROOT/container/migrate/run/unique_id* directory:

- *LocalOldDS.zip*

  A full backup of the previous Teamcenter Integration Framework datastore.

- *LocalNewDS_initial.zip*

  A snapshot of the new Teamcenter Integration Framework datastore with initial files.

- *LocalNewDS_updated.zip*

  A snapshot of the new Teamcenter Integration Framework datastore with initial files and migrated files from the previous datastore.

- *LocalNewDS_final.zip*

  A full backup of the completely migrated datastore.

The following migration-related files are stored in the *TcIF_ROOT/container/migrate/run/unique_id* directory:

- *LocalFilesToDrop.txt*

  A list of files that were not migrated from the previous Teamcenter Integration Framework datastore because they are not required by the updated Teamcenter Integration Framework version.

- *LocalFilesToOverlay.zip*

  The files that were migrated from the previous Teamcenter Integration Framework datastore, regardless of whether the same files also existed in the updated Teamcenter Integration

Framework datastore by default. Preserving these files ensures that several files that typically hold customized values are carried over in the updated datastore.

- *LocalFilesToUpload.zip*

  The files that were migrated from the previous Teamcenter Integration Framework datastore because they did not exist in the updated Teamcenter Integration Framework datastore by default.

- *LocalFilesWithConflict.zip*

  The files that were not migrated from the previous Teamcenter Integration Framework datastore because the same-named files existed in the updated Teamcenter Integration Framework datastore by default.

- *LocalFilesWithConflictList.txt*

  The list of files in *LocalFilesWithConflict.zip* for easy reviewing of the conflict list.

Cluster Teamcenter Integration Framework installation:

A Teamcenter Integration Framework instance that is part of a cluster also has a local data store to use if the instance leaves the cluster in the future. Therefore, the migration involves both the local datastore and the cluster datastore.

The following files are backed up in the *TcIF_ROOT/container/migrate/run/unique_id* directory:

- *LocalOldDS.zip*

  A full backup of the previous Teamcenter Integration Framework local datastore.

- *LocalNewDS_initial.zip*

  A snapshot of the new Teamcenter Integration Framework local datastore with initial files.

- *LocalNewDS_updated.zip*

  A snapshot of the new Teamcenter Integration Framework local datastore with initial files and migrated files from the previous local datastore.

- *LocalNewDS_final.zip*

  A full backup of the completely migrated local datastore.

- *ClusterOldDS.zip*

  A full backup of the previous Teamcenter Integration Framework cluster datastore.

- *ClusterNewDS_initial.zip*

  A snapshot of the new Teamcenter Integration Framework cluster datastore with initial files.

- *ClusterNewDS_updated.zip*

  A snapshot of the new Teamcenter Integration Framework cluster datastore with initial files and migrated files from the previous cluster datastore.

- *ClusterNewDS_final.zip*

  A full backup of the completely migrated cluster datastore.

The following migration-related files are stored in the *TcIF_ROOT/container/migrate/run/unique_id* directory:

- *LocalFilesToDrop.txt*

  A list of files that were not migrated from the previous Teamcenter Integration Framework local datastore because they are not required by the updated Teamcenter Integration Framework version.

- *LocalFilesToOverlay.zip*

  The files that were migrated from the previous Teamcenter Integration Framework local datastore regardless of whether the same files also existed in the updated Teamcenter Integration Framework local datastore by default. Preserving these files ensures that several files that typically hold customized values are carried over in the updated local datastore.

- *LocalFilesToUpload.zip*

  The files that were migrated from the previous Teamcenter Integration Framework local datastore because they did not exist in the updated Teamcenter Integration Framework local datastore by default.

- *LocalFilesWithConflict.zip*

  The files that were not migrated from the previous Teamcenter Integration Framework local datastore because the same-named files existed in the updated Teamcenter Integration Framework local datastore by default.

- *LocalFilesWithConflictList.txt*

  The list of files in *LocalFilesWithConflict.zip* for easy reviewing of the conflict list.

- *ClusterFilesToDrop.txt*

A list of files that were not migrated from the previous Teamcenter Integration Framework cluster datastore because they are not required by the updated Teamcenter Integration Framework version.

- *ClusterFilesToOverlay.zip*

  The files that were migrated from the previous Teamcenter Integration Framework cluster datastore regardless of whether the same files also existed in the updated Teamcenter Integration Framework cluster datastore by default. Preserving these files ensures that several files that typically hold customized values are carried over in the updated cluster datastore.

- *ClusterFilesToUpload.zip*

  The files that were migrated from the previous Teamcenter Integration Framework cluster datastore because they did not exist in the updated Teamcenter Integration Framework cluster datastore by default.

- *ClusterFilesWithConflict.zip*

  The files that were not migrated from the previous Teamcenter Integration Framework cluster datastore because the same-named files existed in the updated Teamcenter Integration Framework cluster datastore by default.

- *ClusterFilesWithConflictList.txt*

  The list of files in *ClusterFilesWithConflict.zip* for easy reviewing of the conflict list.

Manage migration conflicts

No manual resolution is required for files delivered with Teamcenter Integration Framework (unless you have modified these files). That is, files such as localization files from the previous version may have fewer entries compared to the same localization files in the updated version due to new changes, and script APIs may have changed between versions. Files of these types are identified as conflicts during the migration, but you can ignore them unless you have previously customized them.

A running Teamcenter Integration Framework has only one datastore active. The active datastore is the local datastore if Teamcenter Integration Framework is running as a standalone instance. The active datastore is the cluster datastore if Teamcenter Integration Framework is running as part of a cluster. Resolve conflicts for the active datastore at your site.

Use the following general process to resolve conflicts:

1. Evaluate the scope of conflicts by reviewing the *LocalFilesWithConflictList.txt* or *ClusterFilesWithConflictList.txt* summaries.

2. Review the files contained in *LocalFilesWithConflict.zip* or *ClusterFilesWithConflict.zip* to determine which files from the old datastore should be migrated to the new datastore.

3. Back up *LocalFilesWithConflict.zip* or *ClusterFilesWithConflict.zip*. Edit the contents of the *.zip* files such that it contains only those files that should be migrated to the new datastore.

4. Place the edited version of *LocalFilesWithConflict.zip* or *ClusterFilesWithConflict.zip* in the */autoinstall* directory to automatically upload those files to the new datastore.

Resolving unrecoverable migration issues

If an unrecoverable error occurs during migration, the migration stops and a message describing the migration failure appears in the console window. Details of the error are also entered in the log file.

Stop Teamcenter Integration Framework, review the details in the error log, address the issue causing the error, and restart Teamcenter Integration Framework.

Update custom bundles

When upgrading Teamcenter Integration Framework, you may need to manually update custom bundles as follows.

- Groovy scripts that use unsupported APIs will fail. Ensure your scripts are using **the current APIs**.
- If the *platformExtension.jar* file is embedded in a bundle, it must be updated to the most recent version to support API changes.
- Ensure manifest file references are not specific to a version or bundle-version. Use open-ended ranges.
- If third-party libraries are embedded in a bundle, ensure they are updated as necessary. Embedded third-party libraries take precedence over libraries delivered with Teamcenter Integration Framework.
- When upgrading from a version of Teamcenter Integration Framework earlier than 14.0, move any custom bundles from the *TcIF_ROOT/container/bundles* directory to the *TcIF_ROOT/container/deploy* directory. Existing custom *feature.xml* files are ignored and may be removed.

Retain business object definitions

The database schema business object definition (BOD) changed for Teamcenter Integration Framework 10.1.5 and later versions. With this change, you must use the following steps both before and after using Teamcenter Environment Manager (TEM) to apply the patch.

1. Start Teamcenter Integration Framework and open the **Datastore Configuration** web page.

2. Download the following files, retaining the datastore directory structure:

   - **/config/BOSBox.jaxb**
   - **/config/SSOPseudoAppIDMap.jaxb**

- All jaxb files directly under the **/bos** location
- All *connector-config_site-ID*.**jaxb** files directly under the **/config** location
- All *site-type_site-ID*.**jaxb** files directly under the **/sites** location

3. Create an archive (ZIP) file containing all the downloaded files, maintaining the directory structure.
4. Stop Teamcenter Integration Framework.
5. Rename the **datastore.h2.db** file under *TcIF_ROOT***/tcif/container/h2/** to **datastore-***current-release-version***-h2.db**.
6. Start TEM and apply the patch update.
7. Delete all feature XML files that have a previous version in their names under the *TcIF_ROOT***/tcif/container/deploy** directory.
8. Upload the updated files to the Teamcenter Integration Framework datastore. You can do this through the **Datastore Configuration** web page as follows, or you can copy the ZIP file containing the backed-up datastore content into the *TcIF_ROOT***/tcif/container/autoinstall** directory.
   a. Start Teamcenter Integration Framework and open the **Datastore Configuration** web page.
   b. Click **Browse** and choose the ZIP file containing the backed-up datastore content
   c. Select the forward slash (**/**) from the **Select a Location** list and click **Upload**.

Retain JDBC site configurations as PAX-JDBC site configurations

The JDBC connector changed to use PAX-JDBC at Teamcenter Integration Framework 10.1.6. You must retain the previous version JDBC connector configurations as PAX-JDBC configurations.

1. Start Teamcenter Integration Framework and open the site configuration wizard. Obtain the following information about your JDBC sites:
   - **Site ID**
   - **JDBC Driver Name**
   - **Database URL**
2. Stop Teamcenter Integration Framework.
3. In a web browser, search for the following PAX-JDBC properties for your JDBC database:
   - **PAX-JDBC** *driver-name*
   - **PAX-JDBC** *database-name*

4. Start TEM and apply the patch update.

5. Create a data source file as follows and copy it to the *TcIF_ROOT***/tcif/container/etc/** directory.

   Name the file using the following convention: **org.ops4j.datasource-***data-source-name***.cfg**. The file must contain:

   ```
   osgi.jdbc.driver.name=<PAX-JDBC driver name corresponding to JDBC driver used>
       url=<JDBC Database URL>
       dataSourceName=<data sourcename preferred>
       user=<user name for DB connection>
       password=<password for DB connection>
   ```

   > Note:
   >
   > The database user name and password cannot be obtained from the Teamcenter Integration Framework site configuration wizard. You must get this information from the database administrator.

   This makes the data source available to Teamcenter Integration Framework as a data source service.

6. Start Teamcenter Integration Framework and use the site configuration wizard to create a site using the information obtained in step one and the new PAX-JDBC data source you created.

Run Teamcenter Integration Framework

Run Teamcenter Integration Framework from the command prompt

Start the Teamcenter Integration Framework server by entering the following command at a command prompt in the Teamcenter Integration Framework install directory:

```
trun
```

Wait to access Teamcenter Integration Framework until after the "TcIF successfully started" message appears.

Confirm that the server is running by typing the following URL in a browser:

`http://localhost:`*<rest-services-port>*`/tcif/rest/login`

<rest-services-port> is "8090" by default.

Run Teamcenter Integration Framework as a service

Siemens Digital Industries Software recommends using one of the many available commercial or open source service wrappers to run the Teamcenter Integration Framework **trun.bat** script in the *TcIF_ROOT***tcif\container\bin** directory as a service.

Restarting Teamcenter Integration Framework

When **stopping** and restarting Teamcenter Integration Framework or using the **cluster:refresh** command (which restarts Teamcenter Integration Framework), be aware that the operating system coordinates certain subprocesses, including shutting down support processes when the main Teamcenter Integration Framework process is stopped. Under certain circumstances, the shutdown of these services on Windows platforms may not be complete before Teamcenter Integration Framework is restarted, resulting in startup issues.

If startup issues occur, manually shut down Teamcenter Integration Framework and wait several minutes before restarting Teamcenter Integration Framework.

Run Teamcenter Integration Framework instances in a cluster

Multiple Teamcenter Integration Framework instances can run in a clustered mode. A cluster of Teamcenter Integration Framework instances (nodes) work together to process requests in parallel, enhancing the system's overall efficiency and stability. Requests are posted to shared queues to allow requests to be processed by any of the Teamcenter Integration Framework nodes in the cluster.

A cluster differs from a standalone Teamcenter Integration Framework instance in that the cluster uses a shared external database to store configuration information and a shared external database for messaging persistence. The same databases supported by Teamcenter are supported by Teamcenter Integration Framework.

Clustering considerations

Be aware of the following considerations when configuring Teamcenter Integration Framework clusters:

- All nodes in the cluster must be running the same release version of Teamcenter Integration Framework.
- Each Teamcenter Integration Framework node in a cluster must be running on a separate machine. The machines in a cluster must be on the same network and must have unique host names. The machines must be visible to each other so hosts can resolve other hosts by name when interacting.
- An external proxy or load balancer can act as a single point of entry to the Teamcenter Integration Framework cluster.

Configure supporting databases

In the external database used to store configuration information, create a user that can connect and create interactive sessions, has privileges for creating the tables used by Teamcenter Integration Framework, and has a quota for creating tables. Teamcenter Integration Framework creates the tables it needs upon joining the cluster. After the cluster is created, the privileges can be revoked.

Another database or tablespace and user is required for messaging persistence. Apache ActiveMQ is used as the message broker. ActiveMQ creates the tables it needs in the database for persisting messages and clustering. The database used for the ActiveMQ messaging must be of the same type (and use the same driver) as the one used for the Teamcenter Integration Framework tables.

> Note:
>
> When using Teamcenter Integration Framework with Microsoft SQL Server, the first time Apache ActiveMQ connects with the SQL Server database, table modification warning messages are logged. These messages can be safely ignored. The messages begin as follows:
>
> ```
> WARN | Could not create JDBC tables; they could already exist. Failure
> was: ALTER TABLE ACTIVEMQ_ACKS DROP PRIMARY KEY Message: Incorrect
> syntax near the keyword 'PRIMARY'.
> ```
>
> These warnings are due to a known issue with ActiveMQ and SQL Server. The tables are created properly, and the warnings will not be reported in future sessions.

Create or join a Teamcenter Integration Framework cluster

To create a Teamcenter Integration Framework cluster or to add Teamcenter Integration Framework nodes to a cluster, first install Teamcenter Integration Framework as a standalone instance using the steps in **Use TEM to install Teamcenter Integration Framework and (optionally) Teamcenter** or in **Use Deployment Center to install Teamcenter Integration Framework** as appropriate. Once Teamcenter Integration Framework is installed, use the following steps to create or join a cluster:

1. On the machine hosting the standalone Teamcenter Integration Framework installation, open a Teamcenter Integration Framework console window and start Teamcenter Integration Framework using the *trun.bat* batch file in the *TcIF_ROOT\tcif\container\bin* directory. Wait until the console window message stating that Teamcenter Integration Framework has started is displayed before continuing.

2. Ensure all messaging system processing jobs are completed in the Teamcenter Integration Framework instance. When switching from a standalone Teamcenter Integration Framework instance to a cluster node, messages in the instance's messaging system are not migrated to the cluster messaging system.

3. From the open Teamcenter Integration Framework console window, use the **cluster:join** command to add the Teamcenter Integration Framework instance to an existing cluster or to create a new cluster.

The **cluster:join** command takes arguments identifying the type of database to be used and connection information for the database. The command has the following form:

```
cluster:join –activemqDBUrl mq_url –activemqDBUser mq_username –
activemqDBPwd mq_passwd db_type jdbc_url db_username db_passwd
```

where:

- **-activemqDBUrl** *mq_url* is the URL of the messaging persistence database. The URL is required only for the first instance when creating a cluster.
- **-activemqDBUser** *mq_username* is the message database user name. The user name is required only for the first instance when creating a cluster.
- **-activemqDBPwd** *mq_passwd* is the message database user password. The password is required only for the first instance when creating a cluster.
- *db_type* is the type of external database used to store configuration information (**oracle** or **mssql**).
- *jdbc_url* is the JDBC URL for the external database.
- *db_username* is the external database user name.
- *db_passwd* is the external database user password.

Following is an example command for creating a cluster:

```
cluster:join –activemqDBUrl jdbc:oracle:thin:@machinename:1521:amq
 –activemqDBUser scott –activemqDBPwd 12tiger3 oracle
 jdbc:oracle:thin:@machinename:1521:tcifds scott 12tiger3
```

Following is an example command for adding a node to a existing cluster:

```
cluster:join oracle jdbc:oracle:thin:@machinename:1521:tcifds scott
12tiger3
```

The configuration from the instance used to create the cluster is copied onto the cluster. Any configuration information from nodes subsequently joining a cluster is ignored. Therefore, ensure you create the cluster using the instance with your desired configuration.

Enter **help cluster:join** for a full description of the command's syntax.

4. From the Teamcenter Integration Framework console window, use the **cluster:refresh** command to update (refresh) each previously existing node in the cluster after joining a new node to the cluster. (If you have added several nodes to a cluster, perform a refresh on each node in the cluster except

for the last one added.) The refreshes are necessary to synchronize the configuration of previously added nodes to include the latest nodes.

The **cluster:refresh** command has no parameters.

If you encounter restart issues using the **cluster:refresh** command, refer to *Restarting Teamcenter Integration Framework* in **Run Teamcenter Integration Framework**.

Work with Teamcenter Integration Framework clusters

You can perform the following monitoring tasks with Teamcenter Integration Framework clusters:

- List nodes in a cluster

  You can view the current nodes in a cluster using the **cluster:list** command from the Teamcenter Integration Framework console window. The command lists the host names of each node configured as part of the cluster.

  The **cluster:list** command has no parameters.

- Monitor node status

  To determine the operational status of a particular node in a cluster, browse to the following URL:

  ```
  http://host:port/tcif/rest/isAlive
  ```

  where:

  - *host* is the name of the server hosting the node.
  - *port* is the Teamcenter Integration Framework REST service port number.

  For example:

  ```
  http://tcifboston:8090/tcif/rest/isAlive
  ```

  The node responds with its current status.

Remove a Teamcenter Integration Framework node from a cluster

Removing a node from a cluster returns it to being a standalone instance using its embedded H2 SQL database. The messaging server associated with the removed Teamcenter Integration Framework node is no longer part of the cluster and does not share messages with the cluster. Any messages in the messaging system are not migrated to the standalone instance's messaging system. Other nodes in the cluster are still able to process those messages.

Perform the following steps to remove a node from a cluster.

1. From the open Teamcenter Integration Framework console window on the machine hosting the Teamcenter Integration Framework instance to be removed, use the **cluster:leave** command to remove the instance from the cluster.

   The **cluster:leave** command has no parameters.

   Be aware that Teamcenter Integration Framework may report errors when the nodes of the cluster are not properly shut down. The errors are caused by processes not being shut down when Teamcenter Integration Framework is shut down or removed from a cluster improperly. Always use the **cluster:leave** to remove a node from a cluster.

2. Shut down the ActiveMQ server running on the same machine as the removed node by closing the command window.

3. When a node is removed from a cluster, it is shut down. Restart the standalone Teamcenter Integration Framework instance using the *trun.bat* batch file in the *TcIF_ROOT\tcif\container\bin* directory of the machine hosting the instance.

4. The remaining nodes in the cluster must be updated to operate without the removed node as part of the cluster configuration. Run the **cluster:refresh** command once for each remaining node in the cluster.

   The **cluster:refresh** command has no parameters.

   If you encounter restart issues using the **cluster:refresh** command, refer to *Restarting Teamcenter Integration Framework* in **Run Teamcenter Integration Framework**.

Run Teamcenter Integration Framework with Microsoft SQL Server and Java 11

Perform the following steps to use Microsoft SQL Server Teamcenter Integration Framework with Java 11.

1. Download the SQL Server JDBC Driver for Java 11 archive from **www.microsoft.com**. Run the downloaded archive **.exe** file to extract the files from the archive.

2. **Stop Teamcenter Integration Framework**.

3. Copy *mssql-jdbc-7.2.2.jre11.jar* from the extracted files to *TcIF_ROOT\tcif\container\services\statusService*.

   Remove *sqljdbc42.jar* from that directory.

4. Copy *mssql-jdbc-7.2.2.jre11.jar* from the extracted files to *TcIF_ROOT\tcif\container\services\activemq\lib*.

   Remove *sqljdbc42.jar* from that directory.

5. Remove *sqljdbc42.jar* from the *TcIF_ROOT\tcif\container\deploy* directory.

6. Copy the *mssql-jdbc-7.2.2.jre11.jar* from the extracted files to the *TcIF_ROOT\tcif\container\bundles* directory.

7. Start Teamcenter Integration Framework for the changes to take effect.

Migrating from Global Services to Teamcenter Integration Framework

Differences between Global Services and Teamcenter Integration Framework

Teamcenter Integration Framework is based on an OSGi platform and not an application server. Teamcenter Integration Framework differs from Global Services as follows:

- Teamcenter Integration Framework uses OSGi bundles and services instead of enterprise Java beans (EJBs). Instead of an application (EAR file), there are a collection of services.

- The scripting platform changed from BSH to Groovy. Groovy is almost a superset of Java; however, some things do not map directly from Java to Groovy. For example, certain operations with Java generics cause compiler errors. There are some array initializations that must be written differently in Groovy. The scripts distributed with Teamcenter Integration Framework provide good examples.

- The datastore is built into the Teamcenter Integration Framework platform in an H2 database. The datastore starts and stops automatically with the platform. You can change the database used; however, Siemens Digital Industries Software has not tested and does not support Teamcenter Integration Framework using other databases.

- An ActiveMQ JMS system also runs within the Teamcenter Integration Framework platform and is used automatically with the SOAP and REST services. The Java Message Service (JMS) system is set up to automatically retry messages after some small delays and to put the messages into a dead letter queue for potential manual retries. Manual retries repeat the entire process unless you circumvent parts of the process by adding logic to Groovy scripts.

- The Teamcenter Integration Framework security system is simpler than Global Services security and supports single sign-on (SSO).

- The business process language (BPEL) engine used by Global Services is replaced with Groovy scripting in Teamcenter Integration Framework. Because Groovy is similar to Java, it is much easier to write and modify Global Multi-Site (GMS) processes.

- The Global Services message server based on business object definitions (BODs) and reactors is implemented using the Java persistent API (JPA) and a microservice in Teamcenter Integration Framework. The status server microservice listens on an unsecured local port on the Teamcenter Integration Framework machine and has a REST-based API. Teamcenter Integration Framework allows access through a proxy that requires Teamcenter Integration Framework credentials at `http://localhost:8090/tcif/proxy/status/messages`.

- The Teamcenter Integration Framework user interface is extended to improve interaction with the messaging system.

- The Global Services reactor framework is replaced with a Groovy scripting framework in Teamcenter Integration Framework.

- The Global Services Publish servlet is replaced in Teamcenter Integration Framework with either REST services or a new Publish service that supports the same type of URL as the old Publish servlet, but forwards the message to the Groovy scripting framework.

- Global Services web services are supported with minor changes such as the SOAP endpoints. REST services have also been added and are intended to supplant the SOAP services.

- The Global Services schemas are mostly unchanged. However, the auditing schema has many revisions and a new namespace. The connection schema remains unchanged with the exception that the **jndi-name** attribute is renamed to **bean-name**. This impacts BODs you move from Global Services to Teamcenter Integration Framework. The monitoring schema is no longer used along with the query audit process.

- The Teamcenter Integration Framework email service replaces parts of the Global Services notifier reactor and emailing capabilities. It is Camel and velocity based. There is a configuration point and the velocity templates can be edited for customization.

- Teamcenter Integration Framework connector extensions are Groovy scripts. Connector customization can be done through the connector extensions. Rather than listing all of the connector extensions in the connector configuration file, you specify one or more packages and all connector extensions found in the packages are loaded into the connector.

Migration of packages

The main package migrations are:

- The **/com/teamcenter/globalservices/** packages are moved to **/com/teamcenter/esb**.

- The **/com/teamcenter/_globalservices** packages are moved to **/com/teamcenter/esb/internal**. Packages under **/com/teamcenter/esb/internal** are for internal use and not included in the Java documentation.

- Some packages are renamed to fit into the OSGi paradigm to align with the bundles and to avoid splitting packages across bundles. Most of this occurred with the bind bundle and the Java architecture for XML binding (JAXB) classes.

Business object definition templates

A solution may have a BOD that it uses with a particular connector. Teamcenter Integration Framework processes interact with that connector through a site and create the connector configuration for the site automatically in the configuration user interface (UI) at the time the site is created. The configuration user interface fills in the correct connector configuration name into BOD templates found in the datastore. The user interface looks for BOD templates in the *site-type***/*.jaxb** location and, for each of the files, it creates a business object definition in the **/bos** location and fills in the correct connector configuration reference.

Connector configuration templates

When you create a site in the Teamcenter Integration Framework configuration user interface (UI), the UI searches for a template matching the site type of the site. The templates for connector configuration files are stored in the datastore in the **/config/template** location. A server restart may be necessary for changes to templates to take effect.

If the blueprints of the connector specify the site-type attribute as **DevSite**, the template with that site type is used for the site, for example:

```
<box-config name="development" template-name="development" site-
type="DevSite" …>
```

If a template with that site type does not exist, by default, the **CustomConfig** template is used for the site. The connector configuration is created in the **/config** location of the datastore with a name formatted as *connector-config-name_site-id***.jaxb**.

If the **development.jaxb** template has the **DevSite** site-type attribute and the site ID in the wizard is specified as **DevSite**, the wizard creates a **development_DevSite.jaxb** connector configuration file. In BOD data source specifications, the **config-name** attribute must be set to **development_DevSite** to use the connector with that site. If a template for the connector configuration file does not exist, the wizard creates a **CustomConfig_mySite.jaxb** connector configuration file and the **config-name** attribute value must be **CustomConfig_mySite**.

Web services URLs

The **BOSService** web services URL for Teamcenter Integration Framework is **http://localhost:8080/tcif/BOSService?wsdl**. This is for logon, logout, query, and other BOS related web services. This replaces the Global Services URL. The process service for handling various SOAP services has two endpoints: **http://localhost:8080/tcif/process** and **http://localhost:8080/tcif/processAsync**. These replace the **http://localhost:8080/tcgs-ws** and **http://localhost:8080/tcgs-ode** in TcGS URLs in Global Services.

Migrate a Global Services datastore to Teamcenter Integration Framework

The auditing business object definitions (BODs) are not supported in Teamcenter Integration Framework. The Teamcenter Integration Framework configuration user interface provides the interactions with the message service information. The status server REST APIs and JSON can be used as an alternative.

To migrate other BODs, the **conn:data-source-spec** elements:

```
<conn:data-source-spec JNDI-name="TeamcenterSOABox" config-
name="TeamcenterSoaConfig"
```

must be changed to:

```
<conn:data-source-spec bean-name="com.teamcenter.esb.connector.tcsoa"
 config-name="TeamcenterSoaConfig_987654"
```

The **bean-name** attribute value is the connector bundle name.

The **config-name** attribute identifies the target site for the BOD and is formed by appending the site ID to the configuration name for the template.

The following components/files are not used by Teamcenter Integration Framework, so you cannot migrate them:

- The message server configuration files.
- Global Services connector configuration files. Use the configuration user interface to generate all connector configuration files in Teamcenter Integration Framework.
- The notifier reactor. Teamcenter Integration Framework uses Velocity templates for email notifications. Any changes that you make to Global Services email templates you must make in the Velocity templates.
- Global Services **globalservices.properties** file. Use **Configuration→Properties** in the configuration user interface to set these type of properties.
- Global Services security configuration files. Teamcenter Integration Framework security credentials are configured through the configuration user interface and are stored in the datastore.

You can migrate any changes that you make to the **SOASavedQueryMapping** configuration files directly without any modifications.

Mapping control files supplied out-of-the-box (OOTB) with Global Services cannot be migrated. Teamcenter Integration Framework uses a version of the mapping engine that provides improved performance and supplies updated versions of the OOTB control files.

Global Services to Teamcenter Integration Framework published class mapping

Some of the published Global Services components are mapped into Teamcenter Integration Framework bundles. Only bundled public exported resources are identified.

com.teamcenter.esb.bind

Provides binding services for the framework

Exported packages

com.teamcenter.esb.model.bod
com.teamcenter.esb.model.config
com.teamcenter.esb.model.connection
com.teamcenter.esb.model.data
com.teamcenter.esb.model.exception
com.teamcenter.esb.model.form
com.teamcenter.esb.model.security
com.teamcenter.esb.model.table
com.teamcenter.esb.model.util
com.teamcenter.esb.model.valuemap
com.teamcenter.esb.model.where

> Note:
>
> The generated package names retain **globalservices** in the package name. Changing these requires existing systems to also change the way they call Teamcenter Integration Framework. The end system does not see any difference in access.

com.teamcenter.globalservices.audit._2006_12
com.teamcenter.globalservices.bod._2006_12
com.teamcenter.globalservices.config._2006_12
com.teamcenter.globalservices.config.site._2010_06
com.teamcenter.globalservices.connection._2006_12
com.teamcenter.globalservices.data._2006_12
com.teamcenter.globalservices.datastore._2007_06
com.teamcenter.globalservices.failure._2011_06
com.teamcenter.globalservices.form._2006_12
com.teamcenter.globalservices.mapper._2007_06
com.teamcenter.globalservices.menu._2010_06
com.teamcenter.globalservices.monitoring._2007_06
com.teamcenter.globalservices.namespace._2006_12
com.teamcenter.globalservices.process._2007_06
com.teamcenter.globalservices.rule._2006_12
com.teamcenter.globalservices.security._2006_12
com.teamcenter.globalservices.sitemap._2007_06
com.teamcenter.globalservices.table._2006_12
com.teamcenter.globalservices.transfer._2007_06

com.teamcenter.globalservices.util._2006_12
com.teamcenter.globalservices.valuemap._2006_12
com.teamcenter.globalservices.webservice._2006_12
com.teamcenter.globalservices.wsutil._2006_12

Converted packages

com.teamcenter.globalservices.bod
com.teamcenter.globalservices.config
com.teamcenter.globalservices.connection
com.teamcenter.globalservices.data
com.teamcenter.globalservices.exception
com.teamcenter.globalservices.form
com.teamcenter.globalservices.security
com.teamcenter.globalservices.table
com.teamcenter.globalservices.util
com.teamcenter.globalservices.valuemap
com.teamcenter.globalservices.where

com.teamcenter.esb.bos

Provides business object server (BOS) services to the framework.

There are no exported or converted packages.

com.teamcenter.esb.cache

Provides cache functions to the framework.

There are no exported packages.

Converted packages

com.teamcenter._globalservices.cache

com.teamcenter.esb.camel

Provides Camel processes and services. It is the infrastructure for message routing.

There are no exported or converted packages.

com.teamcenter.esb.client

Provides Global Services client functions to the framework.

Exported packages

com.teamcenter.esb.client

Converted packages

com.teamcenter.globalservices.client

com.teamcenter.esb.commons

Provides common utilities to the framework.

Exported packages

com.teamcenter.esb.commons.data
com.teamcenter.esb.commons.exception

Converted packages

com.teamcenter.globalservices.util

com.teamcenter.esb.config

Provides connection configuration services to the framework.

Exported packages

com.teamcenter.esb.config.connection

Converted packages

com.teamcenter.globalservices.config.connection

com.teamcenter.esb.connector

Provides the base connector classes to the framework.

Exported packages

com.teamcenter.esb.connection

Converted packages

com.teamcenter.globalservices.connection

com.teamcenter.esb.connector.jdbc

Provides the JDBC connector.

There are no exported or converted packages.

com.teamcenter.esb.connector.tcsoa

Provides the SOA connector.

Exported packages

com.teamcenter.esb.connection.tc.soa

Converted packages

com.teamcenter.globalservices.connection.tc.soa

com.teamcenter.esb.connector.tcent

Provides the Teamcenter Enterprise connector.

Exported packages

com.teamcenter.esb.connection.tcent

Converted packages

com.teamcenter.globalservices.connection.tcent

com.teamcenter.esb.core

Contains core Teamcenter Integration Framework functions, such as exception handling, messaging, settings, and XML functions.

Exported packages

com.teamcenter.esb.exception
com.teamcenter.esb.msg

Converted packages

com.teamcenter.globalservices.exception
com.teamcenter.globalservices.msg

com.teamcenter.esb.datastore

Contains the Teamcenter Integration Framework datastore.

There are no exported or converted packages.

com.teamcenter.esb.datastore

Contains the Teamcenter Integration Framework datastore.

There are no exported or converted packages.

com.teamcenter.esb.logger

Contains the Teamcenter Integration Framework logging function.

There are not exported packages.

Converted packages

com.teamcenter.globalservices.logger

com.teamcenter.esb.parser

Contains the Teamcenter Integration Framework parsing functions.

There are no exported or converted packages.

com.teamcenter.esb.security

Contains the Teamcenter Integration Framework security functions.

Exported packages

com.teamcenter.esb.security

Converted packages

com.teamcenter.globalservices.security

com.teamcenter.esb.tcsoa

Provides dependent JAR files for the SOA connector.

Exported packages

com.fnd0.schemas.wproxy._2014_10.proxylink
com.fnd0.services.strong.wproxy
com.fnd0.services.strong.wproxy._2014_10
com.teamcenter.net.tcserverproxy.admin
com.teamcenter.net.tcserverproxy.client
com.teamcenter.schemas.core._2006_03.datamanagement
com.teamcenter.schemas.core._2006_03.filemanagement
com.teamcenter.schemas.core._2006_03.reservation
com.teamcenter.schemas.core._2006_03.session
com.teamcenter.schemas.core._2007_01.datamanagement
com.teamcenter.schemas.core._2007_01.filemanagement
com.teamcenter.schemas.core._2007_01.managedrelations
com.teamcenter.schemas.core._2007_01.session
com.teamcenter.schemas.core._2007_06.datamanagement
com.teamcenter.schemas.core._2007_06.lov
com.teamcenter.schemas.core._2007_06.propdescriptor
com.teamcenter.schemas.core._2007_06.session
com.teamcenter.schemas.core._2007_09.datamanagement
com.teamcenter.schemas.core._2007_09.projectlevelsecurity
com.teamcenter.schemas.core._2007_12.datamanagement
com.teamcenter.schemas.core._2007_12.session
com.teamcenter.schemas.core._2008_03.session
com.teamcenter.schemas.core._2008_05.datamanagement
com.teamcenter.schemas.core._2008_06.datamanagement

com.teamcenter.schemas.core._2008_06.dispatchermanagement
com.teamcenter.schemas.core._2008_06.managedrelations
com.teamcenter.schemas.core._2008_06.propdescriptor
com.teamcenter.schemas.core._2008_06.reservation
com.teamcenter.schemas.core._2008_06.session
com.teamcenter.schemas.core._2008_06.structuremanagement
com.teamcenter.schemas.core._2009_04.projectlevelsecurity
com.teamcenter.schemas.core._2009_04.session
com.teamcenter.schemas.core._2009_10.datamanagement
com.teamcenter.schemas.core._2009_10.projectlevelsecurity
com.teamcenter.schemas.core._2010_04.datamanagement
com.teamcenter.schemas.core._2010_04.languageinformation
com.teamcenter.schemas.core._2010_04.session
com.teamcenter.schemas.core._2010_09.datamanagement
com.teamcenter.schemas.core._2011_06.datamanagement
com.teamcenter.schemas.core._2011_06.envelope
com.teamcenter.schemas.core._2011_06.lov
com.teamcenter.schemas.core._2011_06.operationdescriptor
com.teamcenter.schemas.core._2011_06.propdescriptor
com.teamcenter.schemas.core._2011_06.reservation
com.teamcenter.schemas.core._2011_06.session
com.teamcenter.schemas.core._2012_02.datamanagement
com.teamcenter.schemas.core._2012_02.operationdescriptor
com.teamcenter.schemas.core._2012_02.session
com.teamcenter.schemas.core._2012_09.datamanagement
com.teamcenter.schemas.core._2012_09.projectlevelsecurity
com.teamcenter.schemas.core._2012_10.datamanagement
com.teamcenter.schemas.core._2013_05.datamanagement
com.teamcenter.schemas.core._2013_05.lov
com.teamcenter.schemas.core._2014_10.datamanagement
com.teamcenter.schemas.globalmultisite._2007_06.importexport
com.teamcenter.schemas.globalmultisite._2007_06.sitereservation
com.teamcenter.schemas.globalmultisite._2007_12.importexport
com.teamcenter.schemas.globalmultisite._2008_06.importexport
com.teamcenter.schemas.globalmultisite._2010_04.importexport
com.teamcenter.schemas.globalmultisite._2011_06.importexport
com.teamcenter.schemas.multisite._2014_10.importexporttcxml
com.teamcenter.schemas.query._2006_03.savedquery
com.teamcenter.schemas.query._2007_01.savedquery
com.teamcenter.schemas.query._2007_06.finder
com.teamcenter.schemas.query._2007_06.savedquery
com.teamcenter.schemas.query._2007_09.savedquery
com.teamcenter.schemas.query._2008_06.savedquery
com.teamcenter.schemas.query._2010_04.savedquery
com.teamcenter.schemas.query._2010_09.savedquery
com.teamcenter.schemas.query._2013_05.savedquery
com.teamcenter.schemas.soa._2006_03.base
com.teamcenter.schemas.soa._2006_03.exceptions
com.teamcenter.schemas.soa._2006_09.clientcontext

com.teamcenter.schemas.soa._2011_06.metamodel
com.teamcenter.schemas.soa.objectpropertypolicy
com.teamcenter.schemas.workflow._2007_06.workflow
com.teamcenter.schemas.workflow._2008_06.workflow
com.teamcenter.schemas.workflow._2010_09.workflow
com.teamcenter.schemas.workflow._2013_05.workflow
com.teamcenter.schemas.workflow._2014_10.workflow
com.teamcenter.services.loose.core
com.teamcenter.services.loose.core._2006_03
com.teamcenter.services.loose.core._2007_01
com.teamcenter.services.loose.core._2007_06
com.teamcenter.services.loose.core._2007_12
com.teamcenter.services.loose.core._2008_03
com.teamcenter.services.loose.core._2008_06
com.teamcenter.services.loose.core._2009_04
com.teamcenter.services.loose.core._2010_04
com.teamcenter.services.loose.core._2011_06
com.teamcenter.services.loose.core._2012_02
com.teamcenter.services.strong.core
com.teamcenter.services.strong.core._2006_03
com.teamcenter.services.strong.core._2007_01
com.teamcenter.services.strong.core._2007_06
com.teamcenter.services.strong.core._2007_09
com.teamcenter.services.strong.core._2007_12
com.teamcenter.services.strong.core._2008_03
com.teamcenter.services.strong.core._2008_05
com.teamcenter.services.strong.core._2008_06
com.teamcenter.services.strong.core._2009_04
com.teamcenter.services.strong.core._2009_10
com.teamcenter.services.strong.core._2010_04
com.teamcenter.services.strong.core._2010_09
com.teamcenter.services.strong.core._2011_06
com.teamcenter.services.strong.core._2012_02
com.teamcenter.services.strong.core._2012_09
com.teamcenter.services.strong.core._2013_05
com.teamcenter.services.strong.core._2014_10
com.teamcenter.services.strong.globalmultisite
com.teamcenter.services.strong.globalmultisite._2007_06
com.teamcenter.services.strong.globalmultisite._2007_12
com.teamcenter.services.strong.globalmultisite._2008_06
com.teamcenter.services.strong.globalmultisite._2010_04
com.teamcenter.services.strong.globalmultisite._2011_06
com.teamcenter.services.strong.multisite
com.teamcenter.services.strong.multisite._2014_10
com.teamcenter.services.strong.query
com.teamcenter.services.strong.query._2006_03
com.teamcenter.services.strong.query._2007_01
com.teamcenter.services.strong.query._2007_06
com.teamcenter.services.strong.query._2007_09

com.teamcenter.services.strong.query._2008_06
com.teamcenter.services.strong.query._2010_04
com.teamcenter.services.strong.query._2010_09
com.teamcenter.services.strong.query._2013_05
com.teamcenter.services.strong.workflow
com.teamcenter.services.strong.workflow._2007_06
com.teamcenter.services.strong.workflow._2008_06
com.teamcenter.services.strong.workflow._2010_09
com.teamcenter.services.strong.workflow._2013_05
com.teamcenter.services.strong.workflow._2014_10
com.teamcenter.soa
com.teamcenter.soa.client
com.teamcenter.soa.client.model
com.teamcenter.soa.client.model.strong
com.teamcenter.soa.common
com.teamcenter.soa.common.utils
com.teamcenter.soa.exceptions
org.apache.xml.serialize
org.apache.xml.serializer
org.w3._2001.xmlschema

com.teamcenter.esb.services

Provides Teamcenter Integration Framework services.

Exported packages

com.teamcenter.esb.mapper

com.teamcenter.esb.publish

com.teamcenter.esb.service

com.teamcenter.esb.service.util

Converted packages

com.teamcenter.globalservices.service

com.teamcenter.fms

Provides Teamcenter File Services dependencies.

Exported packages

com.teamcenter.fms.servercache
com.teamcenter.fms.servercache. proxy

There are no converted packages.

Migrate a Global Services connector to Teamcenter Integration Framework

Follow the process for how to **add a custom connector to Teamcenter Integration Framework**. The only part of the Global Services connector required is the **ConnectionBoxBean** implementation class.

1. Remove all of the EJB methods from the class (**ejbCreate**, **ejbPassivate**, and so forth). Also remove **throws RemoteException** from the class.

2. Move the **ejbCreate** method into the constructor of the connection box bean implementation.

3. Replace the Global Services packages with the Teamcenter Integration Framework packages in the import statements and anywhere else they appear.

Migrate a solution from Global Services to Teamcenter Integration Framework

This example shows how the Global Services Substance Compliance solution was migrated to Teamcenter Integration Framework. Substance Compliance contained a reactor and supporting classes.

The **com.tccpmreactor.TCCPMReactorBean** class maps to **com.teamcenter.subscmpl.internal.service.CPMEventProcessor** class. The **CPMEventProcessor** class implements the **com.teamcenter.esb.publish.PublishEventHandler** class. **PublishEventHandler** is a Teamcenter Integration Framework component that supports legacy Publish/Reactor solutions.

The SOA connector extensions (**ExtractTcXML** and **ImportTcXML**) are mapped directly to the following Groovy scripts:

- **\script\com\teamcenter\esb\connector\tc\soa\ExtractTcXML.groovy**

- **\script\com\teamcenter\esb\connector\tc\soa\ImportTcXML.groovy**

The CPM connector is unchanged except as described in *Migrate a Global Services connector to Teamcenter Integration Framework*.

Create a custom solution bundle

Create an ant module to build the solution bundle that is used in Teamcenter Integration Framework structure. You do not need to publish this JAR file to **out/jars** and therefore you can publish it to a solution specific configuration. Any additional bundles required to run your solution that are not part of the standard Teamcenter Integration Framework installation must be in the following format.

JAR name	**com.teamcenter.***solution***.solution-11.1.0.jar**
Contents (location in JAR)	**resources/***solutionName***.properties** **META-INF/MANIFEST.MF**

	OSGI-INF\blueprint\bundle-context-osgi.xml **OSGI-INF\blueprint\bundle-context.xml** **com\teamcenter\..*.class**
I18N Contents	**resources/***solutionName***.properties** *your-teamcenter-package-prefix***/internal/msg/***optional***/TextBundleIDs.class**
Installed Location	*TcIFInstallDir***/tcif/container/bundles**

Datastore components

If your solution requires datastore files, create an Ant module to build a ZIP file containing the solution's datastore files. You can publish the resulting file to a solution specific configuration, that is, there is no need for **tcjars** files. Use the following format.

IP file name	*Solution***DataStore.zip**
Contents	**/***solution***/***solution-name***/script/com/.../*.groovy** **/***solution***/***solution-name***/bos/***BOSName***.xml** **/***solution***/***solution-name***/config/***ConfigName***.xml**

Kit the feature file

Source location	**build/kits/kit_tc_cdrom.xml**
Required changes	Locate the other tcif entries in **build/kits/kit_tc_cdrom.xml** file and add the **feature_tcif***solution***intg.xml** to the kit.

Bundle the ZIP file for Teamcenter Environment Manager

The build adds the **TcIFSubsCmplBundle.zip** file to the output. This file contains the **\tcif\container\autoinstall\subsCmplDataStore.zip** ZIP files in the **autoinstall** directory are uploaded to the datastore.

For example, the solution files uploaded for Substance Compliance are:

\solutions\subsCmpl\tcif\container\autoinstall\subsCmplDataStore.zip
\solutions\subsCmp\bos\TeamcenterSoaCommercialPart.jaxb
\solutions\subsCmp\config\TCCPM_solutionConfig.jaxb
\solutions\subsCmp\config\TCCPM_mapping.xsl
\solutions\subsCmp\script\com\teamcenter\esb\connector\tc\soa
\solutions\subsCmp\script\com\teamcenter\esb\connector\tc\soa\ExtractTcXML.groovy
\solutions\subsCmp\script\com\teamcenter\esb\connector\tc\soa\ImportTcXML.groovy

\solutions\subsCmp\tcif\container\bundles\com.teamcenter.subscmpl.solution-13.2.jar

Contains service and solution classes for substance compliance. This replaces the reactor.

\solutions\subsCmp\tcif\container\bundles\tcsoa-subscmpl-fragment-13.2.jar
This fragment adds the required support to the OOTB SOA connector to allow it to call an extension.

Migrate a Global Services reactor to a Teamcenter Integration Framework process

To understand how to convert reactors to processes, refer to how reactors are activated in Global Services and how to activate a process in Teamcenter Integration Framework.

Activate a Global Services reactor

A Global Services reactor responds to a Global Services message that is sent to the Global Services message server. The Global Services message server must be configured through rules to recognize the contents of the Global Services message to determine which Global Services reactor is required to respond. An initiator is used to generate a Global Services message to send to the Global Services message server. An initiator can send a message by calling the Global Services servlet with an HTTP request or by sending a JMS message containing a Global Services message to the queue that the Global Services message server is processing.

In Teamcenter Integration Framework, there is no Global Services message. That message had a very specific format which the Global Services rules engine understood. In Teamcenter Integration Framework, the solution provider determines the best format for the message.

Industry standard methods of handling the messages in Teamcenter Integration Framework

You can create simple JAXB annotated classes and drop them into the datastore to facilitate sending SOAP Messages. Some simple examples of this are shipped in the datastore along with a sample processor. You can add REST endpoints that accept HTTP requests containing the content of the messages as query or path parameters. Alternatively, JSON and other formats can be used with the REST endpoints.

Replacement for initiators

For initiators that used the Global Services publish servlet, a REST endpoint (**PublishService**) was added to Teamcenter Integration Framework that mimics the behavior of the Global Services publish servlet. The service at that endpoint (**http://localhost:8090/tcif/publish**) builds a **PublishEvent** JAXB object and sends it to the Teamcenter Integration Framework message processor, which handles it like a normal request calling a **PublishEventProcessor** interface. That Groovy script either starts another Groovy script registered within it or looks for an OSGi service implementing the **PublishEventHandler** interface with the matching **type** service property. All of the code in the **PublishEventProcessor** interface can be replaced with code to process arbitrary publish events. However, some solutions depend on it calling an OSGi service.

An initiator that sent a Global Services message to a queue for the message server to process must be modified to send a **PublishEvent** object or a JAXB Groovy scripted type. You can also define a Global Services message JAXB Groovy script in Teamcenter Integration Framework and develop a processor for it. However, this is a more complex solution.

If the publish service does not support the HTTP request an initiator sends, that initiator can be directed to (**http://localhost:8090/tcif/rest/publish**) or a similar endpoint. You must write a RESTful Groovy script to handle the HTTP request.

Options for initiating processes within Teamcenter Integration Framework

- Modify the initiator to send and industry standard request (REST/SOAP) and process it in Teamcenter Integration Framework with groovy scripts.
- Use the publish service to accept a Global Services-like URL:

  ```
  [http://localhost:8090/tcif/publish?type=t&publisher=p&class=
  c&source_attr_names=one%5E%5Etwo&sourceattrvalues=bar1%5E%5Ebar2]
  ```

- Write a REST Groovy service to handle a Global Services-like URL.
- Send a JMS message to the ActiveMQ server embedded in Teamcenter Integration Framework with a **PublishEvent** object or a Groovy scripted type.

Migrating reactor code to Teamcenter Integration Framework

Reactors in Global Services extend the **ReactorBean** abstract class and implement the **onMessage** method. The **onMessage** method receives a Global Services message. In Teamcenter Integration Framework, you can annotate a method that has one parameter extended from **JSONObject** or **JAXBBaseObject**, and (optionally) a second parameter that is a **Credentials** object with a **com.teamcenter.esb.service.messaging.JMSListener** annotation. Doing so enables the method to be invoked with the Teamcenter Integration Framework queueing solution. Review the **service.messaging.ProessDispatcher** groovy class in the datastore for an example. See **Using message-oriented middleware solutions and scripting** for more details.

Reactors are meant to respond to an event occurring on a system. Typically, an identifier for the system is in the message. In Teamcenter Integration Framework, the identifier maps to a site ID, allowing the processor to use a connector to retrieve information from the site that initiated the message. If there is only one site where the event can occur, it can be hard coded, but this is not a best practice. The **ProxyService** interface is the best way to interact with other systems from within the processors. It looks up credentials for the site and creates a **BOSClient** instance to interact with the connector. The **ProxyService** interface can also be used to initiate actions on other systems based on the event that occurred on the first system.

Connector extensions are the best way to implement custom logic to interact with the backend connections. For instance, if a JDBC database must be updated based on the message, the **ProxyService** interface can be used to interact with the JDBC connector. This allows access to the JDBC connector extension executed within the context of the JDBC connector.

A processor can call a connector extension for one connector to get data from the first system, call another connector extension on a second connector, and pass the data from the first system to update the second system.

The scope of the **ProcessService** interface code is limited to the class loading context of the Teamcenter Integration Framework services bundle. Therefore, it may be necessary to use **PlatformExtension** bundles to call Groovy scripts that execute within the bundle class loading context with access to Java classes from other software vendors. For example, the services bundle does not have access to the HTML client classes. You must create a platform extending bundle to provide access to those classes.

Notifier reactor functionality in Global Services

The notifier reactor in Global Services receives a Global Services message and uses a rules engine to send an email with the information to a recipient based on the contents of the message. In Teamcenter Integration Framework, there is an email service that you can call with an **EmailRequest** object based on the class from the Global Services schemas.

You configure the Global Services notifier reactor by editing the **config/Notifier.xml** and the **config/EmailTemplate.xml** files in the datastore. In Teamcenter Integration Framework you configure the email service by editing the email templates in the **/templates/*.vm** datastore location. You must also set email properties in the Teamcenter Integration Framework web console. The email templates in Teamcenter Integration Framework are industry standard Apache Velocity templates instead of Global Services-specific templates. The email properties in the Teamcenter Integration Framework user interface are defined by Apache Camel and contain properties like the SMTP server URL.

Migrate a custom BPEL process to Groovy scripts

The GMS processes in Teamcenter Integration Framework are located in the datastore at:

/solution/gms/script/com/teamcenter/globalservices/process/_2007_06

If you have customized GMS BPEL processes to call additional connector extension methods, you can convert the **ConnectorExtension** method to a Groovy script with minor changes. Use the **ConnectorExtension** method of the **ImportObjects** script as an example.

The Groovy scripts are succinct compared to BPEL.

A BPEL assignment XML:

```
<assign> <copy> <from>y</from> <to>x</to> </copy></assign>
```

Converts to a simple assignment in Groovy:

```
x = y;
```

Calling a service:

```
<invoke portType="EmailInterface" inputVariable="email-request-msg"
  outputVariable="email-response-msg">
```

Converts to a simple method call:

```
EmailResponse emailResponse = EmailService.email(emailRequest);
```

JAXB objects and setters replace the XML literal in BPEL. To call a connector extension, make a call to the **ProxyService** interface with a **ProxyServiceRequest** object.

Migrate Global Services email templates to Teamcenter Integration Framework

The substitutable parts in Global Services are formatted as **%gs_transaction_id%**. In Teamcenter Integration Framework, they are formatted as **$body.transaction_id**.

The attributes value from the **param** elements within the **email-request** element are substituted into the template where the name attribute of the **param** element matches the substitution variable name. For example, **$body.transaction_id** gets the value **12345** from the following **email-request.xml** file:

```
email-request
   xmlns="http://teamcenter.com/globalservices/webservice/2006-12"

   xmlns:util="http://teamcenter.com/globalservices/util/2006-12">
  <message-id> data-transfer-success</message-id>
      <to>someuser@xxx.com</to>
        <subject>transfer was successful</subject>
        <util:param name="transaction_id" value="12345"/>
</email-request>
```

> Note:
>
> The **gs_transaction_id** parameter in the templates is renamed to **transaction_id** to show it is no longer specific to Global Services.

3. Configuring and managing Teamcenter Integration Framework operation

Configuring Teamcenter Integration Framework

Teamcenter Integration Framework configuration overview

Configure Teamcenter Integration Framework components using a web browser:

Running the configuration interface

Open Teamcenter Integration Framework configuration in a browser by navigating to a URL having the following format:

`http://`*tcif-server-host*`:`*port*`/tcif/rest/login`

where *tcif-server-host* is your Teamcenter Integration Framework server name, and *port-number* is your Rest Services port (**8090** by default).

The following Teamcenter Integration Framework configuration options are available in the left pane.

Configure

Data Store	Manually update the contents of the Teamcenter Integration Framework datastore.
Properties	Configure general properties of Teamcenter Integration Framework.
Security	Configure user, SSO, and SSL settings.
JDBC Data Sources	View and configure JDBC data sources.
Sites	View, create, edit, and remove sites that participate in Data Exchange.
Activity Status Configuration	Customize the message property columns displayed on **the Activity Status page**.

Queueing

Perform administration tasks related to Teamcenter Integration Framework queues. See **Queueing**.

Activity Status

Search for, view, and delete stored messages. See **Activity Status**.

Data Views

View business object definitions (BODs). See **Business object definitions**.

Documentation

Access Teamcenter Integration Framework API, Java, and other documentation. See **Documentation**.

View Default Log File

Retrieve the current Teamcenter Integration Framework log file. See **View Default Log File**.

Extensions

Provides an extension point where solution management and monitoring can be added to the console. See **Extensions**.

Configure

Data Store

Choose **Configure→Datastore** to update the contents of the Teamcenter Integration Framework datastore.

The datastore contains template files for the service-oriented architecture (SOA) connector, the business object server (BOS) configuration, standard site connectors (JDBC, Teamcenter Enterprise, Teamcenter product master management, Teamcenter), and other standard configurations. For information about the content and format of the datastore configuration files, see *Platform Extensibility Configuration* in the Teamcenter documentation.

The business object definition and most configuration files are Java Architecture for XML Binding (JAXB) files with **.jaxb** extensions. These files allow you to map Java classes to XML representations to store and retrieve data in memory in XML format and are the main content of the datastore. **.jaxb** files use serialization when you upload or download them.

Files with other extensions, for example *.xml* and *.properties* files, are stored in the datastore without being serialized.

Download files from the datastore

The datastore is populated with initial content when Teamcenter Integration Framework is installed.

- View the current directory structure in the **Download** pane.
- Show the contents of a directory by clicking a folder.
- Click a file to download it and save it to a local directory or to open it in the default editor for the file type.

Upload files to the datastore

Use the **Upload** pane to upload a Teamcenter Integration Framework file after you create or modify it.

To upload a group of files, first package the directory as a compressed archive format (JAR or ZIP files). Check **Check to unpack contents of jar/zip file** before uploading the file. Any directories that do not already exist in the datastore are created when you upload the file. Also create new directories by entering the directory name in **Datastore location for object** box. This feature is useful for Teamcenter Integration Framework customizations.

You can disable access to the Upload pane.

Remove files from the datastore

Use the **Remove** pane to remove files no longer needed by Teamcenter Integration Framework. Removing files helps control the size of the datastore database removing obsolete files.

Properties

Properties lists the exposed property bundles (areas) used to configure general properties of Teamcenter Integration Framework. The **display.in.ui** attribute determines whether properties are displayed for the area's configuration. When set to **true**, the property area is visible in the user interface. The **display.in.ui** property must be set in the property file manually. Property files are stored in the *container/etc* directory.

Click ⌄ on an area to see its individual property names and values.

Create a new property area

Display the list of existing property areas and click ⊞ New above the list of areas. Enter a name for the new area and click **OK**. The new area is added to the list.

Property areas map to configuration files in the *tcif/container/etc* directory. The file names have the form `com.tc.esb.`*area_name*`.cfg`. The properties are accessible programmatically using:

```
service.ui.config.ConfigurationProperties.getProperties(String areaName)
```

Edit area properties

1. In the list of property areas, locate the area to edit and click ⌄ to display the area's properties.

2. Update the property values and click **Save changes** to save the changes to the datastore.

3. To create a new property in the area, click ⊞ New to the right of the area name. Enter a name for the new property and click **OK**. Click **Save changes** to save the changes to the datastore.

Security

Principals

Define the attributes that Teamcenter Integration Framework uses to validate user credentials on the **Principals** tab. The **Principals** tab lists currently valid Teamcenter Integration Framework users and lets you perform the following tasks:

Filter the list of Teamcenter Integration Framework users

To refine the list of users, begin entering a string of characters in . As you type, the list updates to show only the users with that string of characters in their names.

Add a new Teamcenter Integration Framework user

1. Click New.

2. Fill in the fields defining the characteristics of the new user. Fields marked with * are required.

Name	Specifies a user name associated with Teamcenter Integration Framework.
Password and Confirm Password	Specifies and confirms the password associated with the user.
Administrator	When set to **True**, specifies the user has Teamcenter Integration Framework administrator privileges.
Directory	Specifies the directory containing business object definitions (BODs) for objects the user can access. When no value is entered for **Directory**, the BODs in the *\bos* directory of the datastore are displayed. To present a different view of the data, specify the name of a directory that is a subdirectory of the *\bos* directory.

3. Click **Create** to add the user to the Teamcenter Integration Framework datastore.

Remove a Teamcenter Integration Framework user

1. Locate the user you wish to remove in the list of Teamcenter Integration Framework users.

2. On the same line as that user's name, click and confirm the removal. The user is removed from the datastore.

Edit the credentials of a Teamcenter Integration Framework user

1. Locate the user you wish to edit in the list of Teamcenter Integration Framework users.

2. On the same line as that user's name, click . The user's characteristics are displayed.

3. Edit the user's characteristics and click **Save** to save the changes to the datastore.

Refresh the list of Teamcenter Integration Framework users

As you work with users, click to update the list of Teamcenter Integration Framework users.

SSO

The **SSO** tab lets you configure Teamcenter Integration Framework for Security Services single sign-on (SSO). Doing so allows a Teamcenter user to log on to any SSO-enabled Teamcenter product and access any other SSO-enabled Teamcenter product using validated credentials.

Specify the following settings on the **SSO** tab. When your changes are complete, click **Save** to commit them to the Teamcenter Integration Framework datastore. Restart the Teamcenter Integration Framework server for the changes to take effect.

Enabled	When checked, specifies that Teamcenter Integration Framework uses Security Services single sign-on.
Admin Attribute	Specifies that this instance of Teamcenter Integration Framework has administrator privileges associated with its user's security credentials.
Application ID	Specifies the unique identifier that Security Services uses to identify this Teamcenter Integration Framework instance. This value must correlate with the IDs entered in the Teamcenter Security Services component's application registries. For information about configuring Teamcenter products in Security Services, see *Security Services Configuration* in the Teamcenter documentation.
Identity URL	Specifies the URL of the Security Services Identity Service, for example: **http:///cvgtsso1:8080/ssoSERVICE**
Login Redirect URL	Specifies the URL to which the client redirects the logon by setting the complete URL for the Security Services logon window, for example: **http://cvgtsso1:8080/ssoLOGINSERVICE/weblogin/login_redirect**
Redirect URLSuffix	Specifies the URL page that Security Services redirects to after a successful logon. This value is appended the Teamcenter Integration Framework redirect URL in Security Services. Typically, you can accept the default **/rest/login** value.
Security Context Decryption Key	Specifies the value used to decrypt a double-encrypted SSO token in cases when context-sensitive security is used. This value must match the value assigned to mediator password parameter when configuring the SSO Login Service. For information about the Security Services mediator password parameter and application tokens, see *Security Services Configuration* in the Teamcenter documentation.

SSL

The **SSL** tab lets you configure Teamcenter Integration Framework for secure encrypted communications using HTTPS.

Specify the following settings on the **SSL** tab. When your changes are complete, click **Save** to commit them to the Teamcenter Integration Framework datastore. Restart the Teamcenter Integration Framework server for the changes to take effect.

Enabled	When checked, specifies that Teamcenter Integration Framework uses an HTTPS encrypted connection.
Key Password	Specifies the password used to protect the private cryptographic key of a public/private key pair.
Keystore	Specifies the location of the repository for the security certificates used for SSL encryption.
Keystore Password	Specifies the password used to access the repository for the security certificates. The Teamcenter Integration Framework security repository has an administration account in its LDAP structure used for maintenance of the repository. Siemens Digital Industries Software recommends that you change the default password value when you first start Teamcenter Integration Framework.
Keystore Type	Specifies the type of keystore you are using. Because Teamcenter Integration Framework is Java-based, the default is JKS. Another commonly used keystore type is PKCS#12, which is not Java-specific. For information about the types of keystores, see the **Java Cryptography Architecture Oracle Providers Documentation** on the Oracle web site.

JDBC Data Sources

JDBC Data Sources lists currently configured data source connections and lets you perform the following tasks:

Filter the list of data source connections

To refine the list of connections, begin entering a string of characters in  . As you type, the list updates to show only the data source connections with that string of characters in their names.

Create a data source connection

1. Click New.
2. Fill in the fields defining the characteristics of the new connection. Fields marked with * are required.

Data Source Name	Specifies a name for this connection.
Database Type	Specifies the type of database to connect to.
JDBC URL	Specifies the URL used to connect to the database.
User	Specifies the name used to log on to the database.
Password	Specifies the database logon password. Confirm the password when prompted.
Pooling	When set to **On**, connection pooling is enabled for this data source.
Pool Size	Specifies the number of connections to cache when pooling is enabled for this data source. The default value is **64**.

3. Click **Create** to add the connection.

Test a data source connection

1. Locate the data source connection you wish to test in the list of connections.

2. On the same line as that connection, click ✔. Teamcenter Integration Framework tests the connection and reports its status along with related details.

Remove a data source connection

1. Locate the connection you wish to remove in the list of data source connections.

2. On the same line as that connection, click 🗑 and confirm the removal. The data source connection is removed.

View and modify existing connection details

1. Locate the data source connection you wish to edit in the list of connections.

2. On the same line as that connection, click ✎. The connection's details are displayed.

3. Edit the connection details. Click **Save** when complete to save the changes.

Sites

Sites allows you to view, create, and edit sites that participate in Data Exchange and to remove site configurations that are no longer participating.

Filter the list of sites

To refine the list of sites, select to filter by ID or type and then begin entering a string of characters in  . As you type, the list updates to show only the sites with that string of characters in their ID or type.

Add a new site

1. Click New.

2. Specify a site ID and type.

ID	Defines the site ID. For Teamcenter sites, the site ID is generated during the installation process and is displayed when you open the site in the Organization application. For JDBC connections, any string is acceptable; a best practice is to provide a string that identifies the target database.
Site Type	Site types are defined by the integrations selected when you installed Teamcenter Integration Framework.

3. Use the **Configuration Parameters** and **Security** tabs to define the characteristics of the new site. Depending on the type of site, parameters will vary.

Security	Defines the Teamcenter Integration Framework users used to access the Teamcenter or Teamcenter Enterprise site. The user name and password must be a valid user at the target site.
JDBC configuration	**Data Source Name** defines the Teamcenter Integration Framework data source used to communicate with the site.
Teamcenter PMM specific configuration	**EndpointURL** Defines the endpoint for the PMM site. **Libraryname** Defines the library that Teamcenter Integration Framework uses to export and import data to the PMM site.
Teamcenter Enterprise configuration	**MUX_HOST** Specifies the name of a computer on which the Teamcenter Enterprise MUX is running. **MUX_PORT** Specifies the port number of a computer on which the Teamcenter Enterprise MUX is running.

DeleteMax

Specifies the maximum number of objects that a user or other client can delete using the delete API. A value of 0 indicates no limit on the number of deletions. The default value is **1**.

db2UITranslation

Specifies whether domain managers convert data between the internal representation and the user interface representation. The default is **false**.

Teamcenter configuration

SOA_URL

Specifies the URL for the Teamcenter SOA service which is the context root for the Teamcenter instance.

CONNECTION_POOL_SIZE

Specifies the maximum number of connections Teamcenter Integration Framework uses to connect to the data source. The default value is **4**.

CONNECTION_POOL_REJUVENATION_RATE

Specifies the number of calls the connector can receive before the current session is ended and new session is started. This parameter is only valid when the associated **CONNECTION_POOL_SIZE** parameter is specified. The default value is **15**.

CONNECTION_MAX_RESERVATION_TIME

Specifies the number of seconds before warnings will be logged about threads with reserved connections to data sources. The default value is **3600**.

4. Click **Create** to add the site.

Remove a site

1. Locate the site you wish to remove in the list of sites.
2. On the same line as that site's name, click and confirm the removal. The site is removed.

Edit a site's configuration

1. Locate the site you wish to edit in the list of sites.
2. On the same line as that site's name, click . The site's characteristics are displayed.
3. Edit the site's characteristics. Click **Save** when complete to save the changes.

Activity Status Configuration

By default, a limited set of message, process, and activity property columns is displayed for search results generated with the **Activity Status** page. If you rely on knowing the values of certain properties regularly, you can add search result columns to display these values. Use the **Activity Status Configuration** page to customize the columns displayed by adding columns for often-needed message properties and by removing unneeded columns.

Add columns

1. On the **Activity Status Configuration** page, locate the property type for which you want to add a column: **Message**, **Process**, or **Activity**.

   If the property exists in the auditing database, select it under **Additional Properties** and drag it to **Custom Columns**.

   If the property does not appear under **Additional Properties**, you can manually create a column for it. Click new for the property type for which you want to add a column, enter the name of the property, and click **OK**. This name will be the column header and must exactly match the property name for values to be displayed in search results.

2. Optionally check **Search** to make the property searchable from the **Activity Status** page. Adjust **size** as necessary to set the column width.

3. Order the columns by dragging them up or down. Columns in search results will be ordered left to right based on the top-down ordering in **Custom Columns**.

4. When you have finished customizing your columns, click **Save**. The columns will be displayed with future search results.

   Click **Cancel** or at any point to cancel your change and to reset the activity status configuration settings to their last saved state.

Remove columns

1. On the **Activity Status Configuration** page, locate the property type for which you want to remove a column: **Message**, **Process**, or **Activity**.

2. Remove columns from search results by dragging them from **Custom Columns** to **Additional Properties**.

3. When you have finished customizing your columns, click **Save**. The columns will be displayed with future search results.

   Click **Cancel** or at any point to cancel your change and to reset the activity status configuration settings to their last saved state.

Update the Additional Properties lists

If the auditing database contains messages with properties not listed as additional properties, you can add these properties to the **Additional Properties** lists without manually adding the individual properties. You can then add columns to search results for these properties.

1. Save any in-process column changes you wish to keep.

2. Click . Then click **OK** to confirm that you want to update the **Additional Properties** lists. Any new properties are added to the lists.

3. Add any of the new properties to the **Custom Columns** lists as necessary.

Queueing

Click **Queueing** to perform the following tasks:

- Create Teamcenter Integration Framework queues
- Monitor Teamcenter Integration Framework queues
- Manage Teamcenter Integration Framework jobs
- Manage Teamcenter Integration Framework queues

Activity Status

Teamcenter Integration Framework generates messages for the activities that it performs. An **ActivityStatus** message object is generated for each process or subprocess (**ProcessStatus** object) and each step (**StepStatus** object). The message state and results are stored in the Teamcenter Integration Framework activity status table in the database. **Activity Status** lets you search for, view, and delete stored messages. You can customize the properties shown for messages.

Search for messages

Search for a particular message by entering its ID in the **Search** field and clicking . The message with that ID is displayed. Use the wildcard characters **%** (matches any number of characters) and **_** (matches a single character) to increase your search results.

For advanced search options, on the same line as the **Search** field, click . Specify your search criteria and click **Search**. A list of messages matching your search criteria is displayed.

Filter the list of messages

To refine the list of messages matching your search criteria, select to filter by ID, type, or user and then begin entering a string of characters in  . As you type, the list updates to show only the messages with that string of characters in their ID, type, or user name.

Delete messages

When deleting messages, you will be prompted to confirm the deletion before any messages are deleted. Use the following techniques to delete messages:

- Delete a particular message in the search results list by clicking on the same line as the message.
- Delete a particular message by entering its ID in the **Delete** field and clicking . Use the wildcard characters **%** (matches any number of characters) and **_** (matches a single character) to specify more messages.
- Delete one or more messages that match particular criteria by clicking on the same line as the **Delete** field. Specify your deletion criteria and click **Delete**.

Data Views

Business object definitions

Search business objects to determine if a particular site configuration is correct and if the business objects are defined correctly in the Teamcenter Integration Framework datastore.

The **Data Views** pane lists the types of business objects defined in the datastore. Click on a listed object type to specify search criteria to use when searching for its objects.

Documentation

Access additional Teamcenter Integration Framework documentation by clicking on **Documentation** in Teamcenter Integration Framework configuration. The following documentation is available:

TcIF API

Displays documentation for the Teamcenter Integration Framework API supported in this release.

Java 11 Doc

Links to the current Java documentation.

Platform

Describes the Teamcenter Integration Framework platform.

Notices & Trademarks

Displays licensing and other information related to Teamcenter Integration Framework.

About

Displays current release information.

View Default Log File

In Teamcenter Integration Framework configuration, click **View Default Log File** to download a *.zip* file containing the current Teamcenter Integration Framework operations log.

Extensions

Extensions provides an extension point where solution management and monitoring can be added to the console.

Extensions references two RESTful Groovy scripts in the datastore. One for configuration (*/solution/extension/script/service/extension/Configure.groovy*) and another for monitoring (*/solution/extension/script/service/extension/Monitor.groovy*). These scripts are completely customizable by downloading the groovy script from the datastore, modifying it, and uploading it back to the same location. Any changes made will be reflected immediately upon refreshing the page in the web browser. The **Path** annotations in the scripts should not be modified or the script will not respond at the proper endpoint.

Configure

Displays the result of the */tcif/rest/extension/configure* page generated by the */solution/extension/script/service/extension/Configure.groovy* script. The script can be edited to generate any desired content, but the intent is to expose configuration of solutions added to Teamcenter Integration Framework which require specialized configuration.

Monitor

Displays the result of the */tcif/rest/extension/monitor* page generated by the */solution/extension/script/service/extension/Monitor.groovy* script. The script can be edited to generate any desired content, but the intent is to expose monitoring of solutions added to Teamcenter Integration Framework.

Configuring SSO users to run Teamcenter Integration Framework configuration

If you have Teamcenter Integration Framework configured for single sign-on (SSO), you must configure one or more Teamcenter Integration Framework user to run **Teamcenter Integration Framework configuration**. Refer to *Configure a Teamcenter Integration Framework administrative user*.

(Earlier versions of Teamcenter Integration Framework supported a more complex configuration model when defining administrative Teamcenter Integration Framework users to support users with Teamcenter Security Services (TcSS) accounts. This configuration model continues to be supported by Teamcenter Integration Framework. Refer to *Legacy SSO configuration approach* when maintaining sites configured using this legacy approach.)

Configure a Teamcenter Integration Framework administrative user

1. For each user, **create a user in the Teamcenter Integration Framework repository** on the **Principals** tab. Create that user with the same name as the corresponding user in the TcSS user security repository.

2. In the Teamcenter Integration Framework repository, assign the user administrative privileges by setting **Administrator** to **True**.

Legacy SSO configuration approach

Earlier versions of Teamcenter Integration Framework required the following approach to configure users with Teamcenter Security Services (TcSS) accounts to also have Teamcenter Integration Framework administrator privileges. This configuration model continues to be supported by Teamcenter Integration Framework.

For ease of maintenance and migration to later releases, you may wish to continue to use the following legacy configuration approach to assign Teamcenter Integration Framework administrator privileges to users with TcSS accounts.

Assign administrator privileges to users with TcSS accounts

1. If you are using the TcSS ApacheDS repository to manage user profile data, use the steps in *Configure ApacheDS for Teamcenter Integration Framework log in* to create a user account attribute that can be used to identify TcSS users as administrators. This same attribute will be used for all TcSS users identified as administrators.

2. If you are using the TcSS-provided ApacheDS repository to manage user profile data, use the steps in *Configure an ApacheDS user as a Teamcenter Integration Framework administrator* to set the newly-created attribute to a value of **True** for each user needing administrator privileges.

   If your site is using any other repository to manage user profile data, such as an existing LDAP repository, use that repository's tools to choose an unused attribute within the user account and set its value to **True**.

   - Choose an attribute you expect will not be needed for another purpose in the future.

   - Use this same attribute for all TcSS users identified as administrators.

3. Disclose the chosen attribute name in Teamcenter Integration Framework:

   a. Run Teamcenter Integration Framework.

   b. Choose **Configure→Security** and click on the **SSO** tab.

c. Enter the attribute name you chose to identify TcSS users as administrators in the **Admin Attribute** field and click **Save**. Be aware the attribute name is case sensitive. See the SSO tab description for additional information.

4.

Configure ApacheDS for Teamcenter Integration Framework log in

1. Open ApacheDS. Using the LDAP Browser, navigate to the following entry:

   dkcou=schema→
   cn=teamcenter→
   ou=attributeTypes

2. Right-click on **ou=attributeType** and choose **New→New Entry**.

3. Keep the default value and click **Next**.

4. In the **New Entry - Object Classes** dialog box, select **metaAttributeType** from the **Available object classes** list. Click **Add** to add **metaAttributeType** to the **Selected object classes** list and then click **Next**.

5. In the **New Entry - Distinguished Name** dialog box, select **m-oid** from the **RDN** list. Set **m-oid** to a unique value such as **1.23.45.6789.0000.1111**.

6. Click **Next** to display the **New Entry - Attributes** dialog box. (Do not click **Finish** on this dialog box at this point.)

7. Click **New Attribute**  , select **m-name** from the **Attribute type** list, and click **Finish**.

8. In the **New Entry - Attributes** dialog box, set **m-name** to a value of **isAdmin**.

9. Click **New Attribute**  , select **m-singleValue** from the **Attribute type** list, and click **Finish**.

10. In the **New Entry - Attributes** dialog box, set **m-singleValue** to a value of **TRUE**.

11. Click **New Attribute**  , select **m-syntax** from the **Attribute type** list, and click **Finish**.

12. In the **New Entry - Attributes** dialog box, set **m-syntax** to a value of **1.3.6.1.4.1.1466.115.121.1.15**.

13. Navigate to the following entry

    ou=schema→
    cn=teamcenter→
    ou=objectClasses

14. Right-click **ou=objectClasses** and choose **New→New Entry**. Keep the default value and click **Next**.

15. In the **New Entry - Object Classes** dialog box, select **metaObjectClass** from the **Available object classes** list. Click **Add** to add **metaObjectClass** to the **Selected object classes** list and then click **Next**.

16. In the **New Entry - Distinguished Name** dialog box, select **m-oid** from the **RDN** list. Set **m-oid** to a unique value such as **1.23.45.6789.0000.2222**.

17. Click **Next** to display the **New Entry - Attributes** dialog box. (Do not click **Finish** on this dialog box at this point.)

18. Click **New Attribute**  , select **m-may** from the **Attribute type** list, and click **Finish**.

19. In the **New Entry - Attributes** dialog box, set **m-may** to a value of **isAdmin**.

20. Click **New Attribute**  , select **m-name** from the **Attribute type** list, and click **Finish**.

21. In the **New Entry - Attributes** dialog box, set **m-name** to a value of **tcif**. (**m-name** can be set to any value, but "**tcif**" describes its association with Teamcenter Integration Framework well.)

22. Click **New Attribute**  , select **m-typeObjectClass** from the **Attribute type** list, and click **Finish**.

23. In the **New Entry - Attributes** dialog box, set **m-typeObjectClass** to a value of **AUXILARY**.

Configure an ApacheDS user as a Teamcenter Integration Framework administrator

1. Using the ApacheDS LDAP Browser, navigate to the following entry

   dc=teamcenter,dc=com→
   ou=users

2. Select the TcSS user to designate as a Teamcenter Integration Framework administrator and click **New Attribute** .

3. Select **objectClass** from the **Attribute type** list and click **Finish**.

4. In the **Edit Entry - Object Classes** dialog box, select **tcif-Admin** from the **Available object classes** list. Click **Add** to add **tcif-Admin** to the **Selected object classes** list and then click **Next**.

5. In the **Edit Entry - Attributes** dialog box, confirm that an **objectClass** attribute with a value of **tcif-Admin** has been added.

6. Click **New Attribute**  , select **isAdmin** from the **Attribute type** list, and click **Finish**.

7. In the **Edit Entry - Attributes** dialog box, set **isAdmin** to a value of **true** and click **Finish**.

Configure the integration framework email service

The Teamcenter Integration Framework email service used by Data Exchange and other processes is configured by the **emailservice.smtp-uri** property in the **com.tc.esb.camel.cfg** configuration file. You can configure the email service in the **Properties** pane of the configuration user interface or directly in the configuration file.

The property must be formatted as a Camel/Spring URI, for example:

```
# Need to supply the smtp server and appropriate from and to.
emailservice.smtp-uri = smtp://tbd.com?from=no-
reply@tbd.com&to=tcifadmin@tbd.com
```

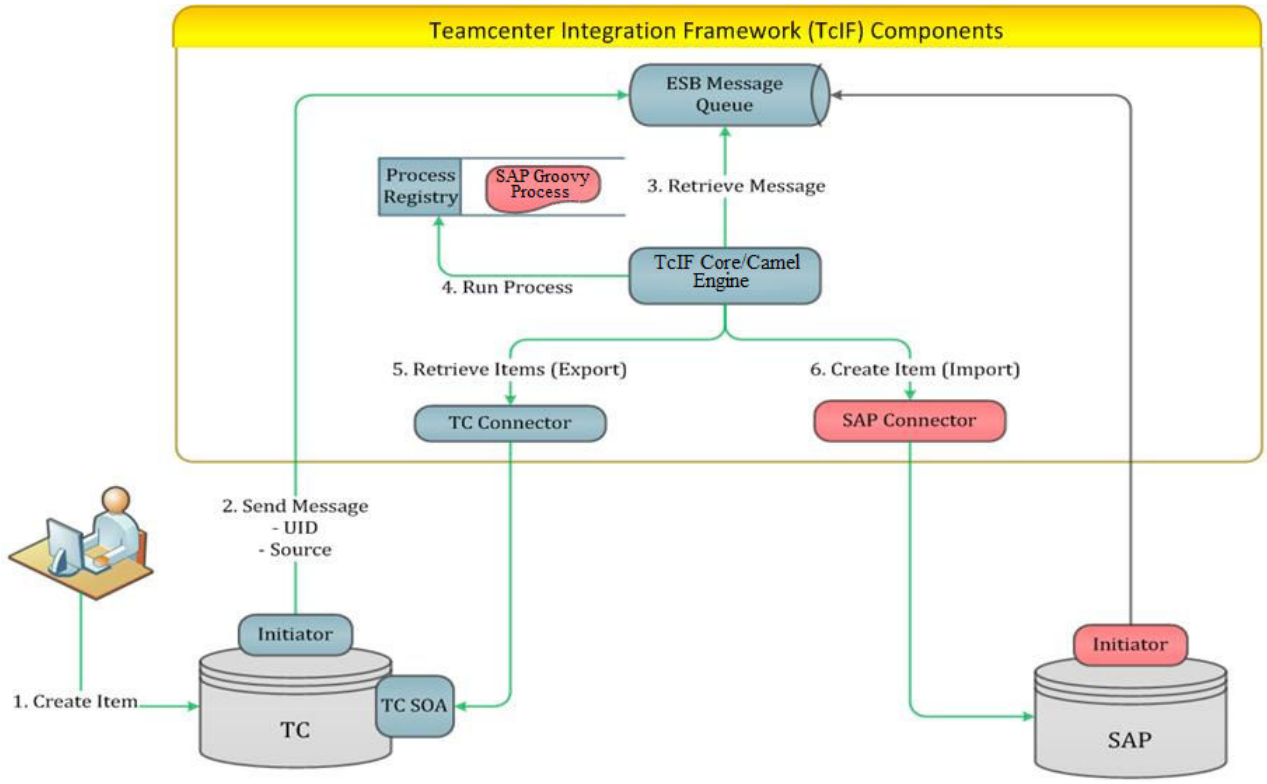
The email service configuration requires that you define an SMTP server and defaults for the message, such as a sending email account and possibly an email account to copy on all messages.

For further information on supported email protocols and parameters, see the Camel web site at:

https://camel.apache.org/

Troubleshoot Teamcenter Integration Framework transfer processes

The following figure shows the transfer flow from between a Teamcenter site and a third-party product for a remote import action. You can use this diagram and the Teamcenter Integration Framework Activity Status user interface to troubleshoot a transfer process.



A diagram showing the transfer of data from Teamcenter to Teamcenter Enterprise would closely mirror the sequence of events in this diagram.

The figure provides the sequence of each step in the process shown. You can view Teamcenter Integration Framework activity status in the configuration interface by choosing **View→Activity status** or in the Teamcenter Integration Framework log file. A Teamcenter Integration Framework activity can have the following statuses:

Active	Activity is in progress.
Active but failing	Activity is in progress and a failure is logged.
Failed	Activity has finished, but failed.
Complete	Activity has finished. No failures logged.

Common activity IDs are:

- **Scheduling**
- **Data Export**
- **Data Import**
- **Data Mapping**
- **Notification**

Control file uploading

To guard against possible uploading of malicious data to the Teamcenter Integration Framework datastore, you can disable access to **the Teamcenter Integration Framework configuration Data Store Upload pane**. By default, the pane is displayed. Hide the pane by setting the `ui.datastore.upload.disabled` parameter in *tcif/container/etc/system.properties* to **true**.

Stop Teamcenter Integration Framework

You can safely shut down and restart any Teamcenter Integration Framework instance without losing any requests. The state of Teamcenter Integration Framework is captured prior to a shutdown, and the instance is brought back to that state when restarted. When Teamcenter Integration Framework is running as a node in a cluster, messages are processed by other instances in the cluster until the restarted node rejoins the cluster.

Stop Teamcenter Integration Framework

Stop Teamcenter Integration Framework from the Teamcenter Integration Framework console window by typing **ctrl-d**.

If Teamcenter Integration Framework is running as a service on Windows, stop the service using the service wrappers which started the service, or by using the Windows Task Manager or Microsoft Management Console.

If you use the operating systems **shutdown -r** command to restart the Teamcenter Integration Framework server, errors may be reported during the restart. These errors are benign and may be ignored.

Teamcenter Integration Framework logging

Teamcenter Integration Framework message objects

The Teamcenter Integration Framework messaging system contains the following objects:

- **LogMessage**

  One **LogMessage** object is created for each unique message received by the system. Objects are uniquely identified by their **MessageID** values.

- **ActivityStatus**

  ActivityStatus provides either a **ProcessStatus** (process or subprocess) or a **StepStatus** (atomic step). An activity may be performed more than once in a process, so each activity is given a unique **activityID** value by the messaging system.

- **ProcessStatus**

  One **ProcessStatus** object is created for each attempt to process a message. Objects are uniquely identified by their **ProcessID** values. A message is normally only processed once; however, if retries are attempted, multiple **ProcessStatus** objects are associated with a **LogMessage** object.

  A process is composed of steps and possibly subprocesses. For instance export, mapping, and import are steps. A replica-to-stub notification is a subprocess that may have one or more steps. A request for a subprocess uses the same **messageID** value as the parent process so it can be tracked as part of the parent process.

- **StepStatus**

  One **StepStatus** object is created for each step in the process. A step is an atomic unit with no substeps. An activity that has multiple steps is tracked by a process status and not a step status. Each of the messaging objects can store additional properties such as client information, logging information, and process state.

  For example, if a process failed at a step, it is possible for that process to track the failure in the messaging system along with the auditing information. If the message is resent, the process can use that information to resume from the point where the previous processing failed.

Configuring Teamcenter Integration Framework exception message logging

Teamcenter Integration Framework uses the Teamcenter Log Manager and a standard message configuration file for exception message logging. You can customize the log file contents by providing a custom configuration file.

In standard Teamcenter Integration Framework, the *TcIF_ROOT\tcif\container\log4j.properties* file controls the logging behavior of log4j.

By modifying the configuration file, you can reconfigure several aspects of the logging process. For example, you can:

- Control how many log files exist, what the log files are named, and where the log files are located
- Set the maximum size of the log file
- Change the format of the entries in the log file
- Set the level of exceptions logged

The properties of the log configuration file define the logging configuration. For information about the logging properties, see the Pax logging documentation at:

https://ops4j1.jira.com

Standard output stream

The first **appender** element in the Teamcenter Integration Framework log configuration file creates a log4j appender object that logs (appends) exception messages to the standard output stream (the **System.out** value on the **param** element).

You can delete this appender if you do not want to log to the standard output stream, or you may want to change the format of the log entries using the **layout** element.

Level definition

The **root** element defines a log4j category that includes all Teamcenter Integration Framework exception messages. You can easily change the type of messages logged by changing the value of the **level** element. For example, if you change the value from **error** to **fatal**, only messages with a fatal level are logged.

Configure Teamcenter Integration Framework tracing in log files

Enable Teamcenter Integration Framework message tracing in the log files to aid you when you are troubleshooting a problem. To enable message tracing:

1. Open the *TcIF_ROOT\tcif\container\log4j.properties* file in a text editor.

2. Locate the **Logging level for all the TciF classes** and **Logging level for TcGS JAXB classes** entries and replace **DEBUG** with **TRACE**.

```
# Logging level for all of the TcIF classes
log4j.logger.com.teamcenter.esb = TRACE
log4j.logger.com.teamcenter.globalservices = TRACE
```

3. Stop the Teamcenter Integration Framework server, remove the log files, and restart the server. Message tracing begins in the new log files once the Teamcenter Integration Framework server restarts.

4. Customizing Teamcenter Integration Framework

Teamcenter Integration Framework examples

Examples with sample code to address different uses of Teamcenter Integration Framework and to simplify Teamcenter Integration Framework development are available in the following locations:

Teamcenter Integration Framework installation directory

TcIF_ROOT\tcif\examples

Unzipped Teamcenter Integration Framework software kit file

TcIF_version\tc\TcIntegrationFramework\tcif\examples

Using Groovy scripts to customize Teamcenter Integration Framework

Groovy scripting environment

Teamcenter Integration Framework supports the use of Java-like classes residing in the datastore. Groovy is a dynamic language for the Java Virtual Machine (JVM) that has additional features, such as closures, builders, and dynamic typing. Although Groovy integrates with all existing Java classes and libraries, certain operations with Java generics cause compiler errors. There are also some array initializations that must be written differently in Groovy.

Groovy provides the ability to statically type check and statically compile your code. Groovy supports domain-specific languages and other compact syntax. The scripts distributed with Teamcenter Integration Framework provide examples.

Unit testing and mocking are supported out-of-the-box and your scripts compile straight to Java bytecode, so you can use it anywhere you can use Java.

When the functionality in the classes in the datastore is accessed, the most recent version of the class uploaded to the datastore is used. No compilation is required outside of the Teamcenter Integration Framework environment. The class can be downloaded from the datastore, modified, uploaded back into the datastore, and the changes take effect immediately. (See the related limitation described at the end of this section.)

The different types of Groovy scripts that can exist in Teamcenter Integration Framework are:

- Scripts that fulfill web service requests.
- Scripts that fulfill REST service requests.

- Scripts that respond to events posted by the **PublishService** service.
- Scripts that extend connectors.
- Scripts that extend the platform.

Most scripts execute within the class loading context of the **com.teamcenter.esb.services** bundle. They have access to the classes that are visible within the services OSGi bundle. Viewing the headers of that bundle provides the complete list of packages that Groovy scripts can access. All but the **ConnectorExtension** and **PlatformExtension** scripts fall in this category.

Similarly, the scripts that extend the connectors can execute within the class loading context of the connector where they are used. These scripts implement the **com.teamcenter.esb.connection.ConnectorExtension** interface and are started using the **execute()** method of the **BOSClient** class. These scripts are intended to interact with the APIs of the system that the connector is interacting with to provide additional capabilities to the connector.

The **ConnectorExtension** scripts are analogous to the **PlatformExtension** scripts. These implement the **com.teamcenter.esb.platform.extension.PlatformExtension** interface. You can create an OSGi bundle with a copy of the **PlatformExtension jar** embedded in it and an Aries Blueprint declaration of an **ExtensionService** spring bean. This structure provides a new class loading context in which groovy scripts can execute. This new OSGi bundle can import packages not accessible by the other contexts where scripts can run. Using the **PlatformExtensionManager** the scripts running in the services bundle can start the platform extension scripts providing access to other packages and versions of software.

Groovy limitation

Scripts can start classes that are declared in other scripts. A Groovy classloader limitation exists where a Groovy class that is called from another Groovy class is not necessarily the latest version uploaded to the datastore. The Groovy classloader does not check to verify the version in the datastore is more recent when a Groovy class is loaded into the Groovy classloader and the following conditions exist:

- The Groovy class is subsequently modified.
- The class is not loaded explicitly by the Groovy classloader (as most Groovy Processes and RESTful scripts are).
- The class is directly referenced from a Groovy class.

This limitation exists because it is not feasible to compile all scripts as they are uploaded as they may require other scripts that have not been uploaded.

Teamcenter Integration Framework processes

Teamcenter Integration Framework acts as an intermediary between Teamcenter and other systems. Typically, when an event in one system needs to trigger a reaction in another system, a handler in the

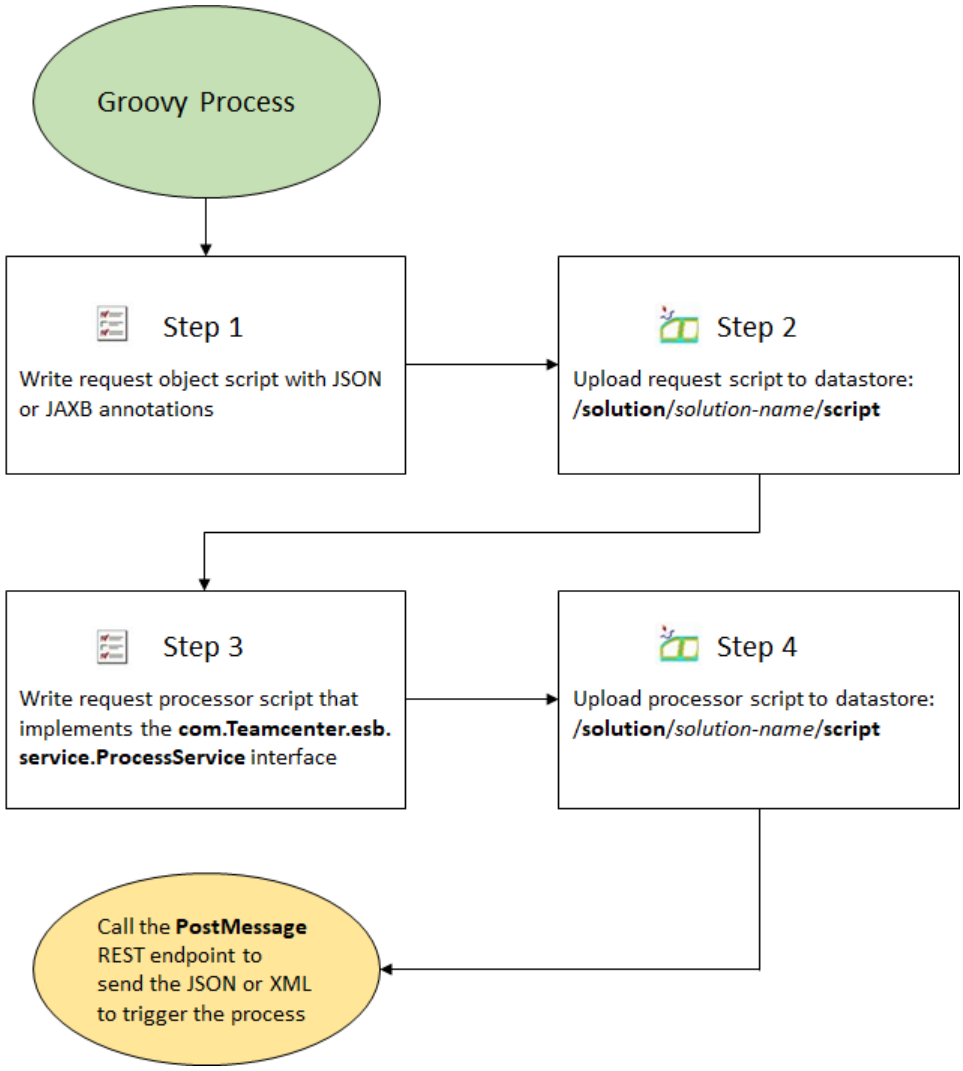
first system communicates that event to Teamcenter Integration Framework using a REST call or SOAP message. (For most scenarios, REST, or JSON over HTTP, are the best choices for this communication.)

Within Teamcenter Integration Framework, the message is pushed on a Java Message Service queue and then given to a processor to handle. Processes are written using Groovy scripts.

If the message is called in a synchronous fashion, a response is returned to the caller. Avoid processes that take long periods of time to complete, as synchronicity can be problematic due to the need to maintain connections and resources.

Creating a Groovy process

Use Groovy scripts to control Teamcenter Integration Framework processes as follows:



Upload the processing Groovy script to the Teamcenter Integration Framework datastore. Call the process by sending a JSON request to the **PostMessage** endpoint (*tcif/rest/messaging/post*) or by sending a SOAP request to the Teamcenter Integration Framework **process** or **processAsync** endpoints.

Requests and responses

All requests and responses are represented by java objects. Two base classes are provided for this purpose: **JSONObject** and **JAXBBaseObject**. These classes override the **toString()** method for converting to JSON and XML, respectively. Response classes are not needed when posting asynchronously or when a string or primitive data type response will suffice. Groovy scripts implementing classes that are sub-classes of these base classes must be uploaded to the data store.

Siemens Digital Industries Software recommends that you place the Groovy file under the */solution/solution-name/script/package-name/* datastore location.

Sending requests

For high volume or resource intensive processes, Siemens Digital Industries Software recommends that you use the **PostMessage** endpoint. This approach automatically takes advantage of the queueing system and the **infrastructure for queueing**, which enables guaranteed delivery, throttling, and timeouts. Within Teamcenter Integration Framework, you can also develop a REST endpoint using a Groovy script that accepts the arguments you want to send and performs the process you want. However, that approach does not leverage the queueing system.

The **PostMessage** endpoint takes a **service.messaging.ProcessRequest** JSON string in the body of the message, for example:

```
{"destination": "queue-name","synchronous": true, "message": "json-or-xml","priority": 4}
```

synchronous and **priority** are optional. Their default values are **true** and **4** respectively. **destination** is optional and generally not used. If **destination** is not specified, either the processor configuration or a rules engine is used to determine to which queue the message is sent.

Queues are associated with particular processors. If a message is of the type expected by a processor associated with a particular queue, that queue is used. Alternatively, queues can have associated properties, and if the message has properties that match those of a particular queue, that queue is used to process the message. If no match is found, a default process queue is used. Therefore, **ProcessMessage** JSON such as `{"message" : "`*message*`"}` is sufficient. (*message* is the JSON or XML message that will be sent to the queue.)

Asynchronous and synchronous web service endpoints are provided which accept arbitrary SOAP requests and include the XML in the body of the SOAP message in the appropriate queue. For SOAP requests, send a web service request with a SOAP body containing XML that corresponds to a Groovy JAXB class in the datastore.

Note:

Siemens Digital Industries Software recommends that you use JSON Groovy objects and the **PostMessage** REST endpoint rather than SOAP and XML. Doing so ensures compatibility with future releases of Teamcenter Integration Framework.

Using the publish servlet or SOAP-based web services may cause Teamcenter Integration Framework to return "Forbidden" or "Unauthorized" HTTP statuses. If you need to access Teamcenter Integration Framework with a session, make the login call normally. Include both the CSRF and session tokens obtained from the login call in subsequent calls to Teamcenter Integration Framework.

JSON requests

Groovy JSON classes must extend the class **com.teamcenter.esb.internal.bind.JSONObject**. The JSON objects have an **@class** field which maps onto a datastore location.

Use the Jackson Databind annotations for controlling the JSON formatting.

Following is an example of a JSON request object script (Upload scripts to */solution/tbd/script/ui/test/ProcRequest.Groovy*):

```
package ui.test;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonInclude.Include;
import com.fasterxml.jackson.annotation.JsonProperty

public class ProcRequest
   extends com.teamcenter.esb.internal.bind.JSONObject {
   public String id_;
   public String type_;

   @JsonProperty("id")
   @JsonInclude(value = Include.ALWAYS)
   public String getID() {
      return id_;
   }
}
```

XML requests

JAXB classes must extend the class **com.teamcenter.esb.internal.bind.JAXBBaseObject**.

Map XML namespaces onto the Groovy JAXB class package names. Map tag names onto class names. For example, if the request XML contains the *http://teamcenter.com/esb/example* namespace attribute value and an **example-request** local name attribute value in the root element, Teamcenter Integration Framework attempts to unmarshal the element into a Groovy JAXB object that is in the */script/com/teamcenter/esb/example/ExampleRequest.Groovy* datastore location. The namespaces are converted

into package names following standard JAXB conventions and the package names are mapped to datastore location names.

The following script includes JAXB annotations that allow Teamcenter Integration Framework to unmarshal it from XML into a Java object. You can convert a schema file into JAXB classes using CXF or Axis2 tools and then change the extension from *.java* to *.groovy* to use it as a Groovy script.

Following is an example of a JAXB request object script:

```
package com.teamcenter.esb.example;

import javax.xml.bind.annotation.*;

@XmlRootElement(name="example-request",namespace="http://teamcenter.com/esb/example")
@XmlAccessorType( XmlAccessType.NONE )
public class ExampleRequest extends com.teamcenter.esb.internal.bind.JAXBBaseObject {
    @XmlAttribute
    private String id_;
    @XmlElement(name="value",namespace="http://teamcenter.com/esb/example")
    private String value_;

    public String getID() {
       return id_;
    }

    public void setID(String id) {
       id_ = id;
    }

    public String getValue() {
       return value_;
    }

    public void setValue(String value) {
       value_ = value;
    }
}
```

Processor scripts

A processor script is a Groovy script that contains a class with one or more methods annotated with the **com.teamcenter.esb.service.messaging.JMSListener** annotation. If there is more than one method, each method must take a different class of argument. The first argument should be a class that extends **JSONObject** or **JAXBBaseObject**. An optional second argument gives the Teamcenter Integration Framework credentials for the user that initiated the request.

If the **JMSListener** annotation has a destination name specified, a queue with that name is created automatically on system startup. Properties of that queue can be specified as well. If the destination name is not specified, the processor is not associated with a queue and you must **create a queue in the Teamcenter Integration Framework web interface** with the processor selected.

If the processor is associated with a queue either in the annotation or in the web interface, and a message is received with the class matching the first argument of the method and no queue specified, then the message is posted in the queue associated with the processor.

Following is an example processor script:

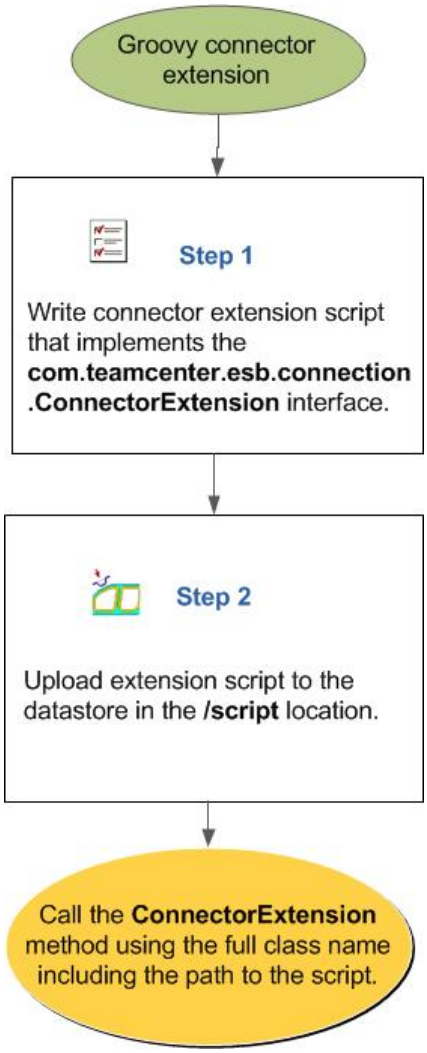
```
package service.ui.test;

import com.teamcenter.esb.model.security.Credentials;

public class FailingProcessor {

   @com.teamcenter.esb.service.messaging.JMSListener
   public static ProcRequest process(ui.test.ProcRequest request) {
      throw new Exception("Failed");
   }

   @JMSListener(destinationName = "test", automaticRetries = 3, stuckTimeOut = 1200000,
                hungTimeOut = 2400000, processorCount = 16, initialRetryDelay = 30000)
   public static Object executeProcess(ui.test.XRequest inputObject, Credentials
credentials)
   throws ESBException {
      return new ProcessManager().executeProcess(inputObject, credentials);
   }
}
```

Status tracking

To track the status of a processor as it is executed on the **Activity Status page** , call the **AuditService.audit()** method prior to each significant activity in the process and then after the activity completes. See the AuditService Javadoc for more information on the process.

On the **Queueing page of the Teamcenter Integration Framework web interface**, the message shows until it is processed successfully.

Creating a custom connector extension using Groovy



A connector extension is a Java or Groovy class that implements the **com.teamcenter.esb.connection.ConnectorExtension** interface. Teamcenter Integration Framework provides some Java connector extension classes out-of-the-box. The framework looks for custom **ConnectorExtension** classes that are groovy scripts uploaded to the Teamcenter Integration Framework datastore in the **/script** location.

The connectors provide query, update, insert, and delete APIs along with an execute method. The execute method is given the name of a method to execute and an array of arguments. The method name is the name of a connector extension provided with the connector or the name of a groovy script implementing the **ConnectorExtension** method in the datastore. You use the full class name including the path for the Groovy script to call the **ConnectorExtension** method.

The following is an example of a connector extension Groovy script:

```
tcesb.test.connector.extension

import com.teamcenter.esb.commons.data.SerializableList;
import com.teamcenter.esb.connection.ConnectionBoxImplementor;
import com.teamcenter.esb.exception.ESBException;

import java.io.Serializable;

class EqualExtension extends com.teamcenter.esb.connection.ConnectorExtension {
   public EqualExtension() {
      // ConnectorExtension constructor takes the name (will not be used for a groovy
script),
      // and the number of arguments expected.  The arguments passed in are checked by
the base
      // connector implementation so that the extension does not have to check.
      super("equal", [1] as Integer[]);
   }

   @Override
   public Serializable execute(String objectName, ConnectionBoxImplementor boxImpl,
      SerializableList<?> argumentList)
      throws ESBException {
      Object argument = argumentList.get(0);
      if (argument instanceof java.lang.String && objectName.equals(argument)) {
         return "equal";
      }
      return "unequal";
   }
}
```

Using message-oriented middleware solutions and scripting

Teamcenter Integration Framework and message-oriented middleware

Teamcenter Integration Framework supports message-oriented middleware solutions and scripting. Leveraging message-oriented middleware can provide several benefits:

- Improved load leveling by letting producers and destinations (queues) send and receive messages at different rates.
- Easier integration and scaling of system resources.
- Decoupled communication lets servers connect as needed and perform their operations in an asynchronous fashion.

By adding the **com.teamcenter.esb.service.messaging.JMSListener** annotation to methods in Groovy scripts, Teamcenter Integration Framework will use the methods to create queues with the specified names or will make the listeners available for later queue creation.

When the listener is associated with a queue, and code in the message's body is associated with a method's Object parameter, Teamcenter Integration Framework can create processing logic that pulls

the message off the queue and calls the method with the message object. See *Using Groovy scripts to customize Teamcenter Integration Framework* for details on creating and working with Groovy scripts.

Create listeners and queues with scripts

Use the following processes to create and deploy Teamcenter Integration Framework queues and listeners with scripts.

Create a listener on the method

Add the **com.teamcenter.esb.service.messaging.JMSListener** annotation to methods in Groovy scripts to let Teamcenter Integration Framework use the methods to create destinations (queues) with the specified names or to make the listeners available for later queue creation. When the listener is associated with a queue, and code (JSON or XML) in the message's body is associated with a method's **Object** parameter, Teamcenter Integration Framework can create processing logic that pulls the message off the queue and calls the method with the message object.

An annotated method must meet one of the following requirements:

- The annotated method must accept one parameter which implements **com.teamcenter.esb.internal.bind.JSONInterface**.
- The annotated method must extend either **com.teamcenter.esb.internal.bind.JSONObject** or **com.teamcenter.esb.internal.bind.JAXBBaseObject**. The method can optionally accept a second parameter, **com.teamcenter.esb.model.security.Credentials**.

The following example creates a queue named "queue-test":

```
package service.ui.test;

public class FailingProcessor {
   @ com.teamcenter.esb.service.messaging.JMSListener(destinationName = "queue-test")
   public static ui.test.ProcRequest process(ui.test.ProcRequest request) {
      throw new Exception("Failed");
   }
}
```

The method returns an instance of any Groovy class that extends **com.teamcenter.esb.internal.bind.JAXBBaseObject** or **com.teamcenter.esb.internal.bind.JSONObject**, or that uses the Java Architecture for XML Binding (JAXB) or Jackson Data-bind (JSON) annotations.

The **com.teamcenter.esb.service.messaging.JMSListener** annotation has options for controlling the processing of messages such as specifying the number of retries and time outs. See the Teamcenter Integration Framework API documentation for details on the **com.teamcenter.esb.service.messaging.JMSListener** annotation's options. (API documentation supporting Teamcenter Integration Framework is available from the Teamcenter Integration Framework web interface. Browse to *<host>*:*<web-ui-port>*/**tcif/rest/login** and click **Documentation**.)

Coded options can also be overridden from the Teamcenter Integration Framework web interface as described in *Manage Teamcenter Integration Framework queues*.

Deploy the script

Deploy Groovy scripts by placing them in the **/solution/***solution-name***/script** directory in the datastore.

For all scripts in that directory that have methods with the **JMSListener** annotation, Teamcenter Integration Framework creates a queue with the name specified on the annotation. If no name is specified on the annotation, then no queue will be created, but the listener will be available for later creation of queues as described in *Manage Teamcenter Integration Framework queues*.

Create Teamcenter Integration Framework queues

Administrators can create Teamcenter Integration Framework destinations (queues) using the Teamcenter Integration Framework web interface. Create a queue from any Teamcenter Integration Framework site in a cluster. Once created and started, the queue is available to all sites in the cluster.

1. Open the Teamcenter Integration Framework web interface by browsing to *<host>*:*<web-ui-port>*/**tcif/controller/index**.

2. In the Teamcenter Integration Framework web interface, click **Queueing** to display the current queues.

3. Click New. The default settings for the queue are displayed.

4. Enter a name for the queue. The queue name must be unique in the Teamcenter Integration Framework cluster.

5. On the **General Settings** tab, update the queue settings as necessary for the queue.

   Stuck Time Out

   Specifies the length of time, in minutes, before an unprocessed job is moved to the stuck state.

   Hung Time Out

   Specifies the length of time, in minutes, a job remains in the stuck state before being moved to the failed state.

   Automatic Retries

   Specifies the number of attempts to make to retry a failed job. If the job does not succeed after this number of retries, it is sent to the dead letter queue (if available).

   Initial Retry Delay

   Specifies the length of time, in milliseconds, to pause before retrying a failed job.

   Dead Letter Queue

When set to **On**, a dead letter queue is available for this queue.

Parallelism

Specifies the number of messages that can be processed simultaneously. Siemens Digital Industries Software recommends setting **Parallelism** to a value of **3** or the total number of processors, whichever is greater.

Setting **Parallelism** to **1** specifies that messages are processed sequentially.

Processor

Specifies the Groovy script that processes messages in the queue.

6. Click **Create**. The queue is created.

7. Locate the new queue in the list of queues. Click  next to the queue name and then click  to start the queue.

Monitor Teamcenter Integration Framework queues

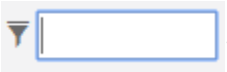
As a Teamcenter Integration Framework user or administrator, you can monitor Teamcenter Integration Framework queues.

To perform the following tasks, first open the Teamcenter Integration Framework web interface by browsing to *<host>*:*<web-ui-port>*/**tcif/rest/login**.

List the current queues

In the Teamcenter Integration Framework web interface, click **Queueing** in the left pane. The currently configured queues are displayed.

Filter the list of queues

To filter the list of queues by name, enter all or part of a queue name in **Filter the Destinations**  .

Refresh the list of queues

Click  to update the list of current queues.

View a queue's settings

Click  for the queue for which you want to view properties. A list of settings such as its status, number of automatic retries, processor, and timeout is displayed.

Click ^ to collapse the setting listing.

Manage Teamcenter Integration Framework jobs

Administrators can manage Teamcenter Integration Framework queues and messages using the Teamcenter Integration Framework web interface.

To perform the following queue management tasks, first open the Teamcenter Integration Framework web interface by browsing to *<host>*:*<web-ui-port>*/**tcif/controller/index**.

List the jobs in a queue

1. In the Teamcenter Integration Framework web interface, click **Queueing** in the left pane. The currently configured queues are displayed.

2. Click for the queue containing the jobs you wish to view. The currently queued jobs are listed along with their current statuses.

Change job states

1. In the Teamcenter Integration Framework web interface, locate the job you want to manage.

2. Change the job state as follows:

   Pause jobs

   Click next to the job name and choose **Pause**.

   Stop jobs

   Click next to the job name and choose **Stop**.

   Advance jobs to the next state

   Click to move the job to the next state.

   Restore jobs in the dead letter queue

   Jobs in the dead letter queue are identified with a symbol. Click to restore a job in the dead letter queue.

   Delete (purge) jobs

   Click to remove a job from the queue. Only jobs in paused or failed states can be purged from a queue.

Manage job properties

Administrators can modify certain properties of queued jobs, such as a job's priority relative to other jobs in the queue, a job's publisher, and a job's time out settings.

1. In the Teamcenter Integration Framework web interface, locate the job you want to manage.
2. Click for the job you wish to view. Detailed information about the job is displayed.
3. Click the **Properties** tab. A listing of the available job properties is displayed.
4. Modify the properties as necessary.
5. Click **Save Changes**. The job is updated with the changes.

Be aware that synchronous messages can be affected by the value of the **camel** area's **request.timeout** property. **request.timeout** specifies the length of time (in milliseconds) the server will wait for a message reply. By default, **request.timeout** is set to `0`, meaning there is no timeout for synchronous messages. Side effects may occur when synchronous messaging requests are made and timeouts are set too low.

The most likely side effect is receiving unexpected responses because processes did not complete in time. Also, messages may remain unprocessed in the reply queue.

Paused synchronous jobs may also show side effects. If a synchronous job is paused, it is moved to another queue. Consequently, the response returned from the server may be the request message that was sent.

If you set **request.timeout** to a value other than `0`, specify a large enough value to avoid possible side effects.

Manage Teamcenter Integration Framework queues

Administrators can manage Teamcenter Integration Framework queues and messages using the Teamcenter Integration Framework web interface.

To perform the following queue management tasks, first open the Teamcenter Integration Framework web interface by browsing to *<host>*:*<web-ui-port>*/**tcif/controller/index**.

Pause and restart a queue

1. Locate the new queue in the list of queues.
2. Review the jobs currently in the queue. Pause any jobs with a pending status.
3. Click next to the queue name and then click Pause to pause the queue. All pending jobs in the queue are paused until the queue is restarted.
4. Click next to the queue name and then click Activate to restart the queue. Pending jobs will be processed in priority order.

Manage queue properties and settings

Administrators can modify certain settings and properties of queues, such as queue priority, the number of job retries, retry intervals, and so on.

1. In the Teamcenter Integration Framework web interface, locate the queue you want to manage and pause the queue.
2. Click ⚙ for the queue you wish to manage. Detailed information about the queue is displayed.
3. Review and update the settings on the **General Settings** and **Properties** tabs as necessary.
4. Click **Save Changes**. The queue is updated with the changes.
5. Restart the queue.

Manage the dead letter queue

Some jobs may not be processed due to the target system being unavailable, network errors, or other environmental issues. After a specified number of failed retry attempts, a job is moved to a storage queue (a dead letter queue) if its target queue has the **Dead Letter Setting** setting enabled. Once issues causing the failure are addressed, the job can be restarted.

In the Teamcenter Integration Framework web interface, locate the job you want to restart. Jobs in the dead letter queue will be listed in their target queue, identified with a 💣 symbol. Click ♻ to restore a job in the dead letter queue.

Delete a queue

1. In the list of queues, locate the queue to be deleted.
2. Let all jobs in the queue complete, or cancel any jobs in the queue.
3. Once the queue is empty of jobs, pause the queue.
4. Click 🗑 for the queue to delete the queue.

Teamcenter Integration Framework properties

Teamcenter Integration Framework is based on an OSGi container with OSGi bundles. The bundles use the Blueprint dependency injection framework to provide configuration properties for the OSGi beans. These properties can be set in the **etc** directory of the container (**tcif/container/etc** directory)

The bundles must register listeners or be restarted for changes to property values to occur at run time. Most of the properties require restart for updates to take effect.

You can expose the properties of any file with the name formatted as **com.tc.esb.***area-name***.cfg** in the **Properties** pane of the user interface. Teamcenter Integration Framework displays the properties in the file if you set the **display.in.ui** property to **true** (**display.in.ui=true** in the **cfg** file).

Some of the standard properties are purposely not set to display in the **Properties** pane because they are not normally changed from the values set during installation.

Track activity status

Use the auditing micro-service in Teamcenter Integration Framework to track activity status. If a Groovy process calls **com.teamcenter.esb.service.AuditClient** with a **com.teamcenter.status.audit.AuditRequest**, **Activity Status** in the Teamcenter Integration Framework configuration interface will display the results. REST calls also can be made to fetch the activity status information. (The endpoint is at *TcIF-host-name***:***rest-port***://micro/status/messages**.)

The **AuditClient** class uses JSON **AuditRequest** objects.

The auditing service creates and updates auditing entries. The auditing information created there can be monitored with **Activity Status**.

Typically, the same audit request is passed to the service many times: once prior to each activity in a process to denote that the activity has started, once after each activity to mark the activity as complete, and once at the end to mark the process complete.

An audit request is composed of three parts.

MessageInfo	Defined once at the start of the process, it describes the message that initiated the process. **MessageInfo** would not be altered by the process, as it is needed for each auditing call to identify the transaction being audited. If a message triggered more than one process, the **MessageInfo** for each of the processes would be the same.
ProcessInfo	Typically defined at the start of the process and marked as completed when the process is done. **ProcessInfo** uniquely identifies the attempts to process the message. Therefore, each attempt should have a unique ID.
ActivityInfo	Replaced or reset at the start of each activity in the process and updated as the activity is processed.

Carefully handle exceptions so that activities are always marked as complete. Otherwise, they may appear as ongoing despite termination of processing due to error conditions. Surround each activity with try/catch blocks. In the catch block, mark the activity as failed and completed. Also be sure to mark the processes completed after the last activity is done.

A typical scenario for auditing a process follows:

1.
```
auditRequest = ProcessUtil.createAuditRequest();
auditRequest.setMessageInfo(ProcessUtil.createMessageInfo(...));
```

```
auditRequest.setProcessInfo(ProcessUtil.createProcessInfo(...));
auditRequest.setActivityInfo(...);
```

2.
```
activityInfo.setActivityId("activity 1");
auditService.audit(auditRequest); // audit start of 1st activity
```

3.
```
ProcessUtil.setActivityToCompleted(...)
auditService.audit(auditRequest); // audit end of 1st activity
```

4.
```
ProcessUtil.setActivityToInProgress(activityInfo);
activityInfo.setActivityId("activity 2");
auditService.audit(auditRequest); // audit start of 2nd activity
```

5.
```
ProcessUtil.setActivityToCompleted(...)
auditService.audit(auditRequest); // audit 2nd activity completed
```

   and so on

6.
```
processInfo.setCompleted(...); // mark process completed
auditRequest.setActivityInfo(null); // not logging activity info in
this case
auditService.audit(auditRequest); // audit end of process
```

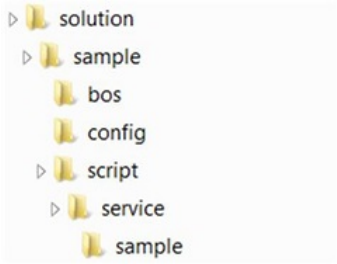
The auditing of the process as completed can be done in conjunction with the auditing call that marks the last activity as completed. That is, using only one call to **auditService.audit()** after marking both the **processInfo** completed and the **activityInfo** completed.

Each of the **MessageInfo**, **ProcessInfo**, and **ActivityInfo** objects has support for additional properties. The additional properties are displayed in the Teamcenter Integration Framework configuration interface and are returned by the REST service.

Teamcenter Integration Framework solution support

Teamcenter Integration Framework supports solutions. A solution typically contains some configuration data and possibly some Groovy scripts. Instead of having those files spread throughout the datastore in the various locations that files are traditionally found, the Teamcenter Integration Framework datastore handles files in solution directories as if they are in the traditional datastore locations.

A solution directory structure starts with the **\solution** directory that contains a directory with the solution name. The solution name directory contains traditional datastore directories as shown in the following example of the structure for a solution named **sample**:



The **sample** solution contains traditional datastore directories including **bos**, **config**, and **script**. The contents of solution directories are handled as if they were in the standard directories with those names, except for precedence rules. For example, the business object definitions (BODs) in the **BOS** directory appear in the data views, the configuration files in the **config** directory are used to configure connectors, and the scripts in the **script** directory are executed. The scripts respond to web service requests, RESTful service requests, connector extensions, and so forth. Scripts in one solution can depend on scripts in other solutions.

To prevent potential problems, use unique names for packages and other content in solutions. There is a defined search order for looking up files across solutions. Files in the standard directory take precedence and then the solutions are searched in alphabetical ordering from A to Z. Therefore, if solution **joe** contains a **/bos/XXX** BOD, that is used instead of the same BOD from solution **sam**, if one exists.

An entire solution can be removed easily by selecting the solution name in the file removal form in the datastore configuration page of the configuration user interface.

Automated solution deployment

The **autoinstall** directory exists in the **container** directory. If you create a ZIP file and move it to the **autoinstall** directory, the contents of the ZIP file are extracted and uploaded to the datastore. You can do the same thing through the user interface on the datastore configuration page. Organize the contents of the ZIP file in directories as described earlier for solutions. That is, use *solution* directory with sub-directories.

Use JAXRS scripts in Teamcenter Integration Framework

Teamcenter Integration Framework supports JAXRS scripting using groovy. You can upload groovy classes containing methods with JAXRS annotations, such as **@PATH** and **@GET** into the Teamcenter Integration Framework datastore. You load them in the **/script/service** directory and its subdirectories. These classes are compiled with the groovy datastore class loader as they are uploaded to the datastore. The methods of the loaded class are inspected to see if they contain the JAXRS PATH annotation. All methods containing the path annotation are exposed as RESTful services on the Teamcenter Integration Framework Representational state transfer (REST) port. The REST port is displayed on the Teamcenter Integration Framework console when you start the framework. The REST port is identified in the **system.properties** file in the **tcif/container/etc** directory.

A best practice is to name the package of the class to match its directory structure under the **script** directory. For example, If you upload a class to the **script/service/test/Hello.groovy** directory in the

datastore, the package name should be **service.test**. If you specify the **@javax.ws.rs.Path("helloworld/you")** annotation on the method, the URL to access that endpoint is **https://tcif-host:rest-port/tcif/rest/helloworld/you**.

You access objects, such as **HttpRequest**, **HttpResponse**, and so forth, using the **@javax.ws.rs.core.Context** annotation. The **@javax.ws.rs.Produces** annotation can be used to specify what the script produces. Class-level JAXRS annotations are not supported. You can pass credentials to the RESTful services with the **j_username** and **j_password** or **TCSSO_APP_USER_ID** and **TCSSO_SESSION_KEY** query parameters.

Teamcenter Integration Framework connectors

Understanding Teamcenter Integration Framework connectors

Teamcenter Integration Framework lets you leverage data stored in systems external to Teamcenter. The framework's connectors define the security and configuration parameters used to communicate between these other data sources (sites) and Teamcenter.

The framework has connectors for Teamcenter, for Teamcenter Enterprise, and for JDBC-enabled data sources (such as Oracle and SQL Server). You can also create custom connectors for other data sources as described in **Configuring and managing connectors**.

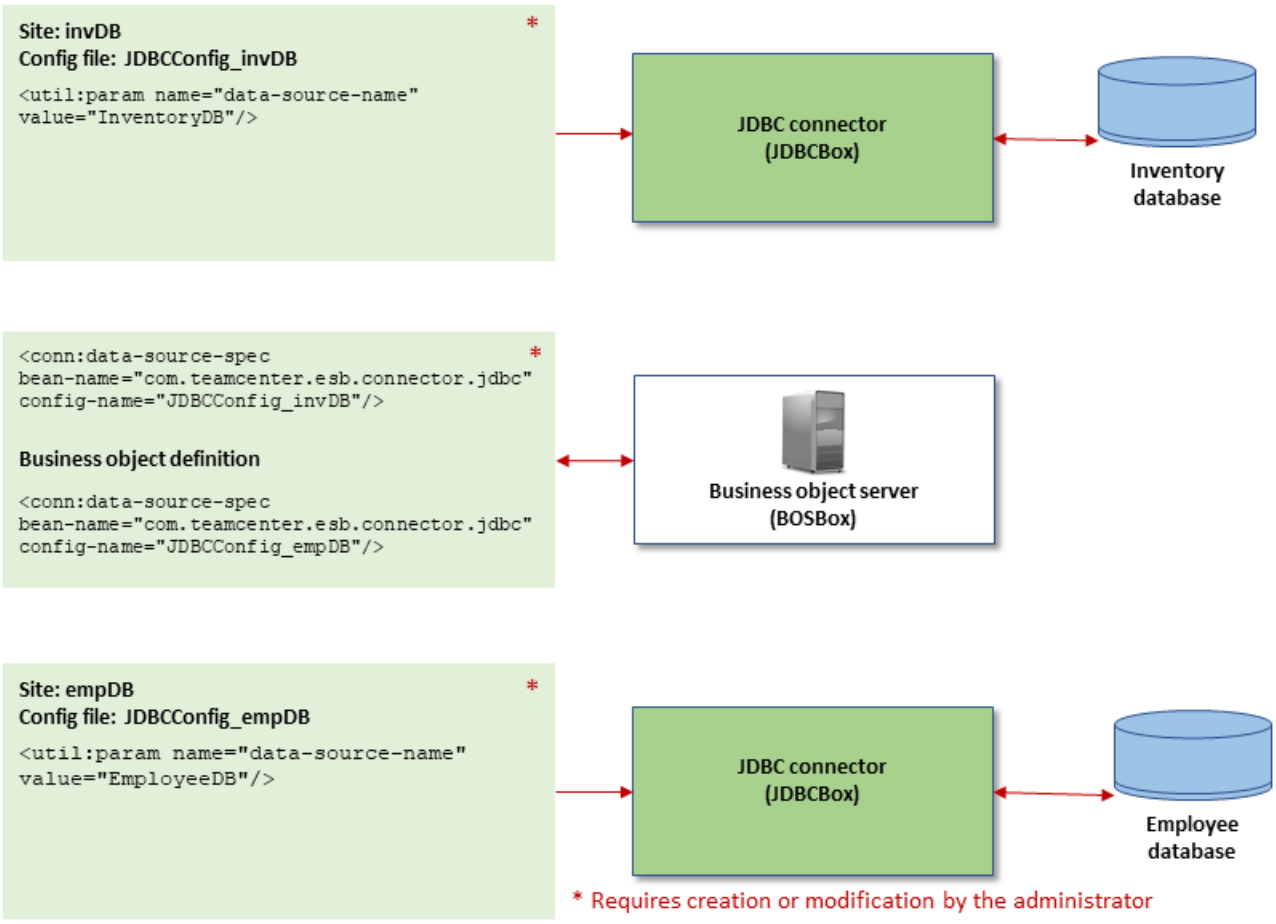
Configuring and managing connectors

The Teamcenter Integration Framework business object server is a connector and requires a connector configuration file.

Connectors are implemented as Spring Beans in OSGI bundles. A spring service is created for the connector factory (**com.teamcenter.esb.connection.ConnectorFactory**). The spring service has 2 properties: **SiteType** and **SiteVersion**. The properties are set for the connector in the blueprint files within the connector bundle. **SiteType** is used for the **Type** in the user interface. (**SiteVersion** is unused.) **ConnectorFactory** is used to obtain an instance of the connector.

When defining a custom connector, you can create a connector configuration template in the datastore where you define the configuration parameters. The parameters in the template are shown on **the Teamcenter Integration Framework configuration Sites page** where the administrator can specify values for each site.

When used, the connector instance is given a **ConnectionBoxConfig** instance with the parameter values from the site and a **Credentials** instance with the security information.



Configuring connectors

Teamcenter Integration Framework connector configuration

Each connector determines the properties that define the information required to communicate with the data source. All connectors contain the following XML elements:

- XML elements common to configuring all Teamcenter Integration Framework connectors. These are defined in the **box-config.xsd** and **util.xsd** schema files located in the **config** folder in the Teamcenter Integration Framework datastore.
- XML elements and attributes for configuring business object server, web services, and JDBC database connectors.

If your enterprise created a custom connector to a data source, you must get the connector properties from the connector's creator and create configuration files using custom XML parameters (**util:param** element) definitions.

Create and manage configuration files

When configuring connectors, manually edit the configuration files of custom connector templates. Perform all other connection configurations using **the Sites page**. Create a site whenever you wish to interact with a backend system of any type. Specify configuration parameters and security credentials using the **Sites** page.

Teamcenter Integration Framework provides sample template files for each type of connector in the */config/template* directory. Use this same directory to templates uploaded for custom connectors.

Use the following procedure to manually create a custom connector configuration template:

1. Create a text file with the **.jaxb** extension.

   If you use an XML editor to create the file, you can specify the **connection.xsd** file to use when the editor requests a schema. Teamcenter Integration Framework validates all configuration files loaded into the datastore. Validating the file before uploading the file to the datastore prevents errors when you upload the file. Find the **connection.xsd** file in the *container/wsdl* directory.

   > Caution:
   >
   > For Teamcenter Integration Framework to work correctly, the Teamcenter Integration Framework configuration file must be valid according to the **connection.xsd** schema. Use a validating editor to avoid encountering errors when you upload the file to the datastore.

2. Name the configuration file. You can provide any name for the configuration file.

   The following box-config element attributes are required for custom connector configuration templates:

   template-name

   Specifies the name of the configuration files generated when you create sites for the custom connector

   site-type

   Matches the **SiteType** attribute of the **Spring** service specified in the blueprints of the custom connector bundle.

   SiteType must be unique in each connector. Therefore, **site-type** is also unique.

   Connector configuration files are created from the template when a site is created and named *<template-name>_<siteID>***.jaxb**. Configuration files are stored in the *config* directory of the datastore.

   Example:

   ```
   <box-config name="JDBCConfig"
     template-name="JDBCConfig"
   ```

```
site-type="JDBC"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="connection.xsd"
xmlns:util="http://teamcenter.com/globalservices/util/2006-12"
xmlns="http://teamcenter.com/globalservices/connection/2006-12">
```

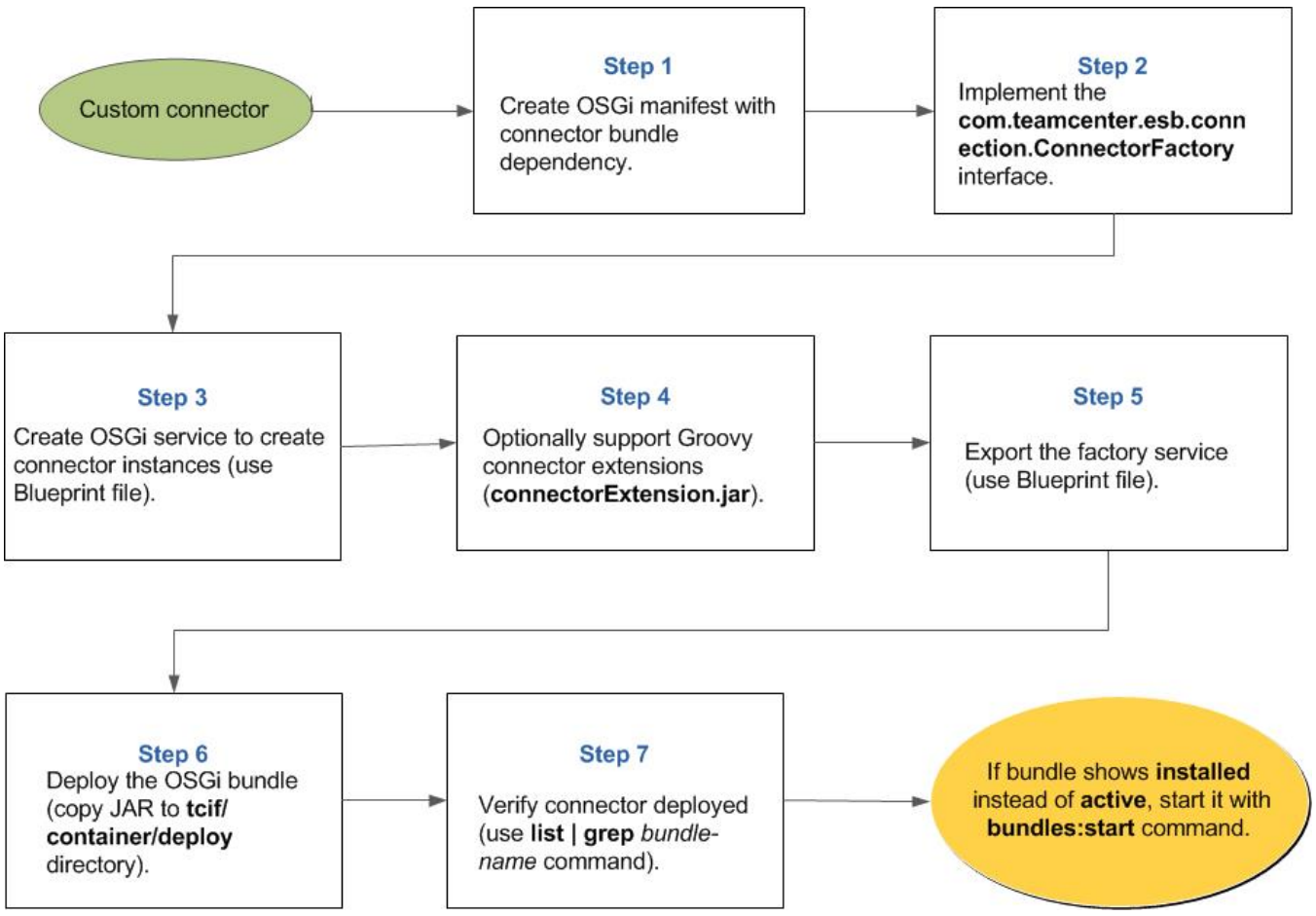
3. Define the data source properties.

4. Save the connector configuration file in a temporary directory.

5. Upload the configuration template file to the */config/template* datastore location.

> Note:
>
> To revise connector configuration files, download the file from the datastore to a temporary directory, modify the elements in the file, and upload the file to the */config/template* datastore location. It is not necessary to stop or restart any application server components.

Add a custom connector to Teamcenter Integration Framework



Teamcenter Integration Framework is based on an OSGI container with OSGI bundles using the Blueprint dependency injection framework. To add a custom connector you must create a new OSGI bundle. There is a maven archetype in the **examples** directory that helps you create custom connectors. It does many of the following steps for you.

1. Create an OSGI manifest file that lists the Teamcenter Integration Framework **Connector** bundle as a dependency. This connector bundle contains the interfaces that your custom connector must implement.

   This is the minimum requirement. Include other bundles as necessary (for example, **Core** and **Binding** bundles).

2. Implement the **com.teamcenter.esb.connection.ConnectorFactory** interface for your connector.

3. Use the Blueprint file to create an OSGI service that implements the connector factory interface to create instances of the connector.

   The connector factory interface is a singleton created when the bundle is deployed. Clients use it to get instances of the custom connector.

4. If your custom connector must support **Groovy connector extensions**, embed the **connectorExtension.jar** file in the custom connector bundle and include the **OSGI-INF/blueprint/datastore-classloader-context.xml** file.

   Add the **connectorExtension.jar** file to the manifest classpath of the connector bundle. Refer to the out-of-the-box Teamcenter Integration Framework connectors as examples.

5. Use the Blueprint files of the custom connector bundle to export the factory service. The following example shows the service properties that enable to configuration user interface to create sites based on a custom connector.

   ```
   <service-properties>
      <entry key="SiteType" value="my-connector-type"/>
      <entry key="SiteVersion" value="my-connector-version"/>
   </service-properties>
   ```

6. Deploy the OSGI bundle (JAR file) for your custom connector by copying the file into the **tcif/container/deploy** directory.

   After it is successfully deployed, the custom connector site type (**my-connector-type**) appears in the list of connectors that the site configuration wizard displays.

   To set configuration parameters on the connector, you can download the file from the data store, edit it, and upload it.

7. Use the follow command to check if the connector successfully deployed:

list -t=0 | grep *bundle-name*

The bundle name is a substring of the *bundle-SymbolicName* value specified in the OSGI custom connector bundle. The command returns a value in the following format showing the connector is deployed:

[279] [Active] [ ] [80]*bundle-name (bundle-version)*

The first number in square brackets is the bundle number. **Active** indicates the bundles state. **Created** appears in the third set of square brackets when the Spring services are created.

To troubleshoot an issue (for example, if **Active** and **Created** are missing when you check the deployment), run the following commands

bundles:headers *bundle-number*

Use the *bundle-number* on the system. If the bundle state is installed, use the following command to start it or determine why it did not start:

bundles:start *bundle-number*

Extend a connector in Teamcenter Integration Framework

In Teamcenter Integration Framework, avoid creating subclasses from the connector implementation classes. Siemens Digital Industries Software recommends that you write **ConnectorExtension** Groovy scripts and upload them to the datastore. The connector extension scripts are then invoked through the **execute** method of the connector. When invoked, the script is passed the instance of **ConnectionBoxImpl** that is associated with the **Connector** upon which the execute method has been invoked. Through the **ConnectionBoxImpl** instance, the back end connection is accessed and operations are performed against it.

If a **ConnectorExtension** script is uploaded to the datastore into the same package as the **ConnectionBoxImpl** class for which it is designed to be used, the connector discovers it and makes it accessible by the **ConnectorExtension** name. The **ConnectorExtension** name is defined in the **ConnectorExtension** constructor as in the following example:

```
package com.teamcenter.esb.connection.tc.soa;
class MyExtensionScript extends ConnectorExtension {
   public MyExtensionScript() {
      // The first parameter is the name of this connector extension
      // operation. The second parameter denotes the number of arguments
      // passed to this connector extension.
      super("MyExtensionScript", 1);
   }
```

The **ConnectorExtension** name is “MyExtensionScript”. Because the script is in the package **com.teamcenter.esb.connection.tc.soa**, **TeamcenterSOABoxImpl** will be aware of the **ConnectorExtension** if it is uploaded into the appropriate directory in the datastore. For example, */solution/mySolution/script/com/teamcenter/esb/connection/tc/soa*.

If **execute(“MyExtensionScript”)** is called against the **TeamcenterSOA** connector, it will invoke the **execute(String, ConnectionBoxImplementor, Object...)** method of the **ConnectorExtension**.

Using specific packages

You can configure each of the standard connectors to use a different package for the connector extensions that the connector binds by name by adding a parameter such as **<parameter name=”connector.extension.package” value=”some.package”/>** in the connector configuration file for the site. This causes the **Connector** to load the **ConnectorExtension** scripts from the specified package rather than from the default package.

Using fully-qualified class names

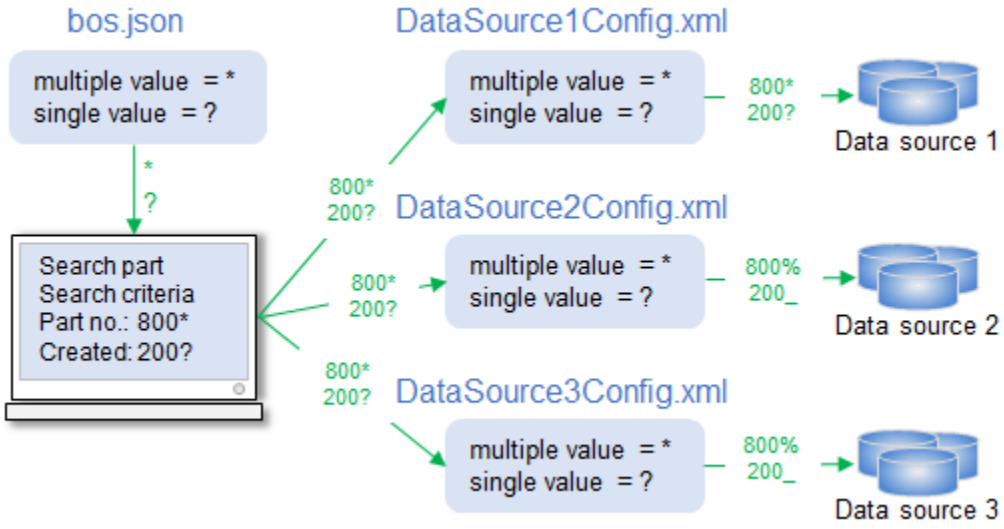
You can specify the fully-qualified class name of the script rather than the **ConnectorExtension** name when calling the execute method of the connector with the form **connector.execute(“my.package.MyExtension”, ...)**. In this example, the **Connector** searches for a **ConnectorExtension** script in the datastore in */solution/*/script/my/package/MyExtension.groovy*, compiles it, and invokes the **execute** method defined in the Groovy script.

Configuring wildcard characters

To accommodate an environment in which users submit one request to multiple data sources that each recognize different wildcard characters, Teamcenter Integration Framework enables you to configure wildcard characters for each data source. The business object server (BOS) converts BOS wildcards into wildcards for the connector when it sends a request to the connector. A connector uses the wildcards defined in its configuration file. If the connector configuration file does not define any wildcard characters, the BOS wildcards are passed to the connector unchanged with the exception that any escape characters are removed.

You define the wildcard the user enters to match the wildcard the data source supports. In the Teamcenter Integration Framework user interface, these configured wildcards are implemented in the search form. If your enterprise has created a custom user interface, they may also be implemented for other operations, such as updating and deleting data.

Wildcard character translation:



The connector configuration file configures the set of wildcard characters users enter and the wildcard escape character described in the following table. The file is stored as a serialized object in the **config** folder in the Teamcenter Integration Framework datastore. To change the defaults, download the file from the datastore to a temporary directory and modify the XML element values. After completing your changes, you must upload the edited file to the **/config** location in the datastore.

The wildcard characters are displayed on the object search form. When you change the characters, the changes automatically display on the form (no need to reload the XML file).

If you do not include the wildcard XML elements in the connector configuration file Teamcenter Integration Framework does not translate the wildcard characters users enter.

Siemens Digital Industries Software recommends using a validating editor to avoid errors due to invalid XML files when modifying a connector configuration file.

Wildcard	Description	Default	XML element
Positional	Character user enters to match any one character. For example, if the positional wildcard is the underscore (_), the user enters the following to search for all strings that begin with 25 and have any single additional character: **25_** Among other 3-character strings, this search could return **250** and **25%**.	_	**single**
String	Character user enters to match zero or more characters.	%	**multiple**

Wildcard	Description	Default	XML element
	For example, if the string wildcard is **%**, the user enters the following to search for all strings that begin with **25**: **25%** Among other strings, this search could return **25**, **25%** and **25,999**.		
Escape	Character user enters to indicate that the character immediately following should be treated literally rather than as a wildcard. For example, if the string wildcard is the percent symbol (%), and the escape character is the backward slant (\), the user enters the following to search for **25%**: **25\%** This search returns **25%** only.	\	**escape**

Data object repository connector

The data object repository connector allows a connection to a JDBC datastore (or any persistent data source) for query, insert, update, and delete of data objects.

To configure the data object repository connector, create a site on **the Sites page** using a site configuration with the type **database**. The connector has no configuration parameters. The connector uses the **DataStoreDataSource** data source, which appears on **the Data Sources page**. View **DataStoreDataSource** on the **Data Sources** page for the user. When creating the site, specify the security parameters to match those set in the data source.

You can modify any BOD to use the connector by changing the **data-source-spec** element to use this connector as follows:

```
<conn:data-source-spec bean-name="com.teamcenter.esb.connector.repository"
config-name="databaseConfig_<siteID>"/
```

If you create a site **dbr** as a **database** type site, take any BOD and set the data source specification to:

```
<conn:data-source-spec bean-
name="com.teamcenter.esb.connector.repository"
config-name="databaseConfig_dbr"/>
```

If the BOD has the **insert-able** parameter set to **true**, you can then perform inserts using the Teamcenter Integration Framework configuration UI into the data object repository for that BOD. Be aware that the connector configuration template for the data object repository has a template-name of **databaseConfig**, resulting in the value for **config-name** in the **data-source-spec** example for a site named **dbr**.

Following is an example of a **PublicationRecord** BOD that uses this connector.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--
bcprt
This software and related documentation are proprietary to Siemens Digital
Industries Software. COPYRIGHT 2005 UGS CORP.  ALL RIGHTS RESERVED
ecprt
-->
<business-object-definition name="PublicationRecord"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:conn="http://teamcenter.com/globalservices/connection/2006-12"
          xsi:noNamespaceSchemaLocation="business-object-definition.xsd"
          xmlns="http://teamcenter.com/globalservices/bod/2006-12">
  <label default="PublicationRecord" ID="PublicationRecord.name"/>
  <data-source name="database">
    <data-segment name="A">
      <conn:data-source-spec bean-name="DORepositoryBox"
                             config-name="DataObjectRepositoryConfig"/>
      <class-name>PublicationRecord</class-name>
      <insert-able>true</insert-able>
      <delete-able>true</delete-able>
      <!--
         - Define the publication record tag to have the
         - same value as the pubr_object_tag.
         -->
      <dynamic-attribute name="pubr_tag">
         <label default="pubr_tag" ID="PublicationRecord.attr.pubr_tag"/>
         <script-text>
            concat $pubr_object_tag ""
         </script-text>
      </dynamic-attribute>
      <attribute name="pubr_object_tag">
        <label default="pubr_object_tag" ID="PublicationRecord.attr.pubr_object_tag"/>
        <primary-key>true</primary-key>
        <required-for-insert>true</required-for-insert>
        <search-able>true</search-able>
        <update-able>false</update-able>
      </attribute>
      <attribute name="pubr_owning_site">
        <label default="pubr_owning_site" ID="PublicationRecord.attr.pubr_owning_site"/>
        <type>int</type>
        <update-able>true</update-able>
      </attribute>
      <attribute name="pubr_pub_date">
        <label default="pubr_pub_date" ID="PublicationRecord.attr.pubr_pub_date"/>
        <update-able>true</update-able>
      </attribute>
      <attribute name="pubr_object_class">
        <label default="pubr_object_class"
ID="PublicationRecord.attr.pubr_object_class"/>
        <update-able>true</update-able>
```

```
        </attribute>
        <attribute name="pubr_object_type">
          <label default="pubr_object_type" ID="PublicationRecord.attr.pubr_object_type"/>
          <update-able>true</update-able>
        </attribute>
        <attribute name="pubr_object_id">
          <label default="pubr_object_id" ID="PublicationRecord.attr.pubr_object_id"/>
          <update-able>true</update-able>
        </attribute>
        <attribute name="pubr_object_name">
          <label default="pubr_object_name" ID="PublicationRecord.attr.pubr_object_name"/>
          <update-able>true</update-able>
        </attribute>
        <attribute name="pubr_owner_id">
          <label default="pubr_owner_id" ID="PublicationRecord.attr.pubr_owner_id"/>
          <update-able>true</update-able>
        </attribute>
        <attribute name="pubr_group_id">
          <label default="pubr_group_id" ID="PublicationRecord.attr.pubr_group_id"/>
          <update-able>true</update-able>
        </attribute>
        <attribute name="pubr_flag">
          <label default="pubr_flag" ID="PublicationRecord.attr.pubr_flag"/>
          <type>int</type>
          <update-able>true</update-able>
        </attribute>
        <attribute name="ods_site">
          <label default="ods_site" ID="PublicationRecord.attr.ods_site"/>
          <type>int</type>
          <update-able>true</update-able>
        </attribute>
        <attribute name="app_ref">
          <label default="app_ref" ID="PublicationRecord.attr.app_ref"/>
          <update-able>true</update-able>
        </attribute>
        <attribute name="release_status">
          <label default="release_status" ID="PublicationRecord.attr.release_status"/>
          <update-able>true</update-able>
        </attribute>
      </data-segment>
    </data-source>
  </business-object-definition>
```

Teamcenter SOA connector configuration elements

SOA connector

The service-oriented architecture (SOA) connector provides a connection to a Teamcenter site using the standard SOA protocol. You configure the SOA connector using **the Sites page** and the procedure described in *Create and manage configuration files*.

The standard SOA connector supports the following functionality:

- Query using Teamcenter saved queries
- Insert for standard item, folder, and dataset objects and their subtypes
- Update

- Delete
- Teamcenter download through FMS
- Teamcenter upload through FMS
- Revise
- Create relation
- Delete relation
- Check out (does not support different reservation types)
- Check in
- Get default group
- Set default group
- Change file owner
- Change object owner

You can configure the SOA connector for client-side connection pooling. Client-side connection pooling limits the number of connections used by Teamcenter Integration Framework to connect to a particular data source, and reuses connections across sessions run by the same user. If a Teamcenter Integration Framework session requires access to a pooled data source connection, it waits until one is available, uses it, and then frees it for use by other sessions.

You can configure saved query attributes to allow the SOA connector to support the Teamcenter database local language for saved queries. The properties used to find and execute a saved query in a local language are defined in one of the following files in the **config** datastore location:

- **SoaSavedQueryMapping_local_***local_id_country***.properties**

- **SoaSavedQueryMapping_local_***local_id***.properties**

- **SoaSavedQueryMapping_local_.properties**

Teamcenter Integration Framework looks first for the **SoaSavedQueryMapping_local_***locale_id_country_id***.properties** file with the locale ID (*locale_id*) and country ID (*country_id*) matching the locale ID and country ID of the Teamcenter Integration Framework server. If this file is not found, Teamcenter Integration Framework looks for the file with the correct locale ID, and finally, for the default **SoaSavedQueryMapping_local_.properties** file.

There are locale-country specific files for each language Teamcenter supports in the **config/templates** datastore location. If you have non-English hosts in your Teamcenter Integration Framework or Teamcenter environment, you must configure the SOA connector for the appropriate language.

Configure SOA connector locale

- If your Teamcenter Integration Framework web application and Teamcenter database are running on hosts using the same locale:

  1. Download the language specific file from the **config/templates** datastore location.

  2. Upload the language specific file to the **config** datastore location.

For example, if both the Teamcenter Integration Framework and Teamcenter database are hosted using the Japanese locale, download the **SoaSavedQueryMapping_locale_ja_JP.properties** file from **config/templates** and upload it to the **config** datastore location.

- If your Teamcenter Integration Framework web application and Teamcenter database are running on hosts using different locales:

  1. Download the language specific file for the Teamcenter database locale from the **config/templates** datastore location.

  2. Rename the language specific file to match the locale ID and country ID of the host running the Teamcenter Integration Framework web application.

  3. Upload the renamed file to the **config** datastore location.

  For example, if your Teamcenter Integration Framework web application is on an German language host and your Teamcenter database is on a Japanese language host, rename the **SoaSavedQueryMapping_locale_ja_JP.properties** file to **SoaSavedQueryMapping_locale_de.properties**.

Custom BOD and saved queries

The SOA connector uses Teamcenter saved queries, which allows it to use the same mechanism that other Teamcenter clients use to obtain Teamcenter query results. The saved query interface works with names of saved query objects and saved query attribute names. These names can be localized, so they are not required to directly match the type and attribute names of the resulting objects. For flexibility, Teamcenter Integration Framework supports a configurable cross-reference table for associating Teamcenter classes with saved queries.

You can create custom saved queries for querying the Teamcenter database for custom business objects. You must add the query properties for the custom saved query to the appropriate language specific **SoaSavedQuerymapping_locale.properties** file. You must provide a set of properties, starting the **saved-query-mapping.** prefix, for each saved query. The following is required for each saved query:

- The property must be unique.
- The **name** property must specify a name of a saved query in the Teamcenter database.
- The **used-for** property must specify the object names used by the saved query. In the BOD, this is equivalent to **class-name** elements. Do not localize this property's values.
- You must provide an **object_type** attribute.

For example, the saved query for Teamcenter items is used for the following Teamcenter objects:

- **Item**

- **TcSoaItem**
- **TeamcenterSoaItem**

Therefore, the **used-for** property entry for Teamcenter items in the saved query mapping file is:

```
saved-query-mapping.item.used-for = Item,TcSoaItem,TeamcenterSoaItem
```

Each attribute mapping has the following format:

saved-query-mapping.*object-name***.attribute.***attribute-name* **=** *attribute-value*

The *attribute-name* is the object's attribute name to map to the saved query attribute. The attribute value is the value of the **name-on-source** element in the BOD, or the **attribute** element's **name** attribute if the attribute does not have a **name-on-source** element in the BOD. The following is the attribute mappings for the Teamcenter **ItemRevision** object:

```
# The Teamcenter object "ItemRevision" has a saved query named "Item Revision..."
saved-query-mapping.itemRev.name = Item Revision...
# The saved query for the Teamcenter "ItemRevision" is used for the Teamcenter
# objects "ItemRevision", "TcSoaItemRevision" and "TeamcenterSoaItemRevision"
# [do not localize the used-for values]
saved-query-mapping.itemRev.used-for = ItemRevision,TcSoaItemRevision,
TeamcenterSoaItemRevision
# The saved query for Item ("Item Revision...") has an attribute "Name" that
# corresponds to the "object_name" attribute from "ItemRevision".
saved-query-mapping.itemRev.attribute.object_name      = Name
# "ItemRevision"."object_type" --> "Item Revision..."."Type"
saved-query-mapping.itemRev.attribute.object_type      = Type
# "ItemRevision"."object_desc" --> "Item Revision..."."Description"
saved-query-mapping.itemRev.attribute.item_id          = Item ID
# "ItemRevision"."object_desc" --> "Item Revision..."."Description"
saved-query-mapping.itemRev.attribute.object_desc      = Description
# "ItemRevision"."user_id" --> "Item Revision..."."Owning User"
saved-query-mapping.itemRev.attribute.user_id          = Owning User
# "ItemRevision"."name" --> "Item Revision..."."Owning Group"
saved-query-mapping.itemRev.attribute.name             = Owning Group
# "ItemRevision"."item_revision_id" --> "Item Revision..."."Revision"
saved-query-mapping.itemRev.attribute.item_revision_id = Revision
```

> Caution:
>
> The localized names must match the localized names in the saved queries exactly. Map only values that are required by the saved query.

The following is the localized attribute mapping for the Teamcenter item revision for the **ja_JP** locale:

```
# The Teamcenter object "ItemRevision" has a saved query named "Item Revision..."
saved-query-mapping.itemRev.name = \u30a2\u30a4\u30c6\u30e0
\u30ea\u30d3\u30b8\u30e7\u30f3...
# The saved query for the Teamcenter "ItemRevision" is used for the Teamcenter
# objects "ItemRevision", "TcSoaItemRevision" and "TeamcenterSoaItemRevision"
```

```
# [do not localize the used-for values]
saved-query-mapping.itemRev.used-for =
ItemRevision,TcSoaItemRevision,TeamcenterSoaItemRevision
# The saved query for Item ("Item Revision...") has an attribute "Name" that
# corresponds to the "object_name" attribute from "ItemRevision".
saved-query-mapping.itemRev.attribute.object_name      = \u540d\u524d
# "ItemRevision"."object_type" --> "Item Revision..."."Type"
saved-query-mapping.itemRev.attribute.object_type      = \u30bf\u30a4\u30d7
# "ItemRevision"."object_desc" --> "Item Revision..."."Description"
saved-query-mapping.itemRev.attribute.item_id          = \u30a2\u30a4\u30c6\u30e0ID
# "ItemRevision"."object_desc" --> "Item Revision..."."Description"
saved-query-mapping.itemRev.attribute.object_desc      = \u8aac\u660e
# "ItemRevision"."user_id" --> "Item Revision..."."Owning User"
saved-query-mapping.itemRev.attribute.user_id          = \u6240\u6709\u8005
# "ItemRevision"."name" --> "Item Revision..."."Owning Group"
saved-query-mapping.itemRev.attribute.name             =
\u6240\u6709\u30b0\u30eb\u30fc\u30d7
# "ItemRevision"."item_revision_id" --> "Item Revision..."."Revision"
saved-query-mapping.itemRev.attribute.item_revision_id = \u30ea\u30d3\u30b8\u30e7\u30f3
```

Note:

The localized names are Unicode (UTF-8) encoded values.

param

PURPOSE

Specifies the parameter names and values used to configure the connector.

SYNTAX

<util:param name="*parameter-name***" value="***parameter-value***" />**

ATTRIBUTES

name

Specifies the name of a parameter. The following parameter names can be used to configure the SOA connector:

Parameter	Description
SSO_enabled	Specifies whether the Teamcenter instance this connector communicates with is using single sign-on (SSO) functionality. Valid values are **true** or **false**.
SSOApplicationID	Specifies the Teamcenter application ID when the instance is SSO enabled.
SOA_URL	Specifies the URL for the Teamcenter SOA service which is the context root for the Teamcenter instance. You must provide an **SOA_URL**parameter.
CONNECTION_POOL_SIZE	Specifies the maximum number of connections Teamcenter Integration Framework uses to connect to the data source. This parameter is optional.
CONNECTION_POOL_ REJUVENATION_RATE	Specifies the number of calls the connector can receive before the current session is ended and new session is started. This parameter is only valid when the associated **CONNECTION_POOL_SIZE** parameter is specified. This parameter is optional and if not specified along with the **CONNECTION_POOL_SIZE** parameter, Teamcenter Integration Framework uses a default value of 15.
RETIRE_CONNECTION_ AFTER_IMPORT	Indicates that Teamcenter Integration Framework terminates the SOA connection after completion of the import process of a transfer. If your users may export multiple versions of an object concurrently, or are importing briefcase files, set this parameter to **TRUE** to avoid potential loss of versions or unexpected stub objects at the imported site.

Parameter	Description
	The default behavior is to terminate the connection (**TRUE**) after the import process completes.

value

Specifies the parameter value. For example:

```
HTTP://localhost:7001/tc
```

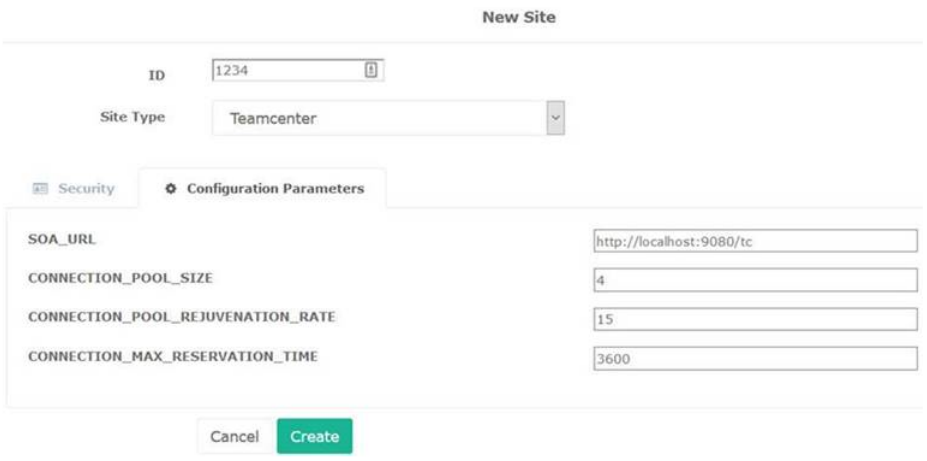
And the **SOA_URL** parameter is:

```
<util:param name="SOA_URL" value="http://localhost:7001/tc"/>
```

You must enter a value for the **SOA_URL** attribute.

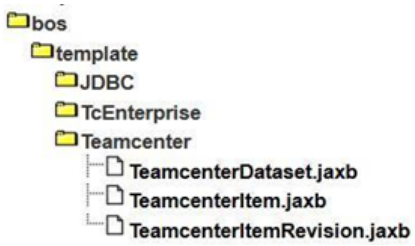
You can configure the SOA connector for client-side connection pooling. Client-side connection pooling limits the number of connections used by Teamcenter Integration Framework in connecting to a particular data source, and reuses connections across sessions run by the same user. This causes the Teamcenter Integration Framework session to wait until a pooled connection is available, uses it, then free it for use by others. If you require client-side pooling, you must set the **CONNECTION_POOL_SIZE** parameter. This setting specifies the maximum number of connections that may be active for this connector configuration. The associated **CONNECTION_POOL_REJUVENATION_RATE** parameter defaults to 15 if you do not specify a value. This setting specifies the number of calls the connector receives before the current session is ended and a new session is started (rejuvenation).

EXAMPLE



SOA business object definitions

Teamcenter Integration Framework provides three templates for SOA BODs as follows. Solutions like Supplier Collaboration may add their own templates as well.



When you create a Teamcenter Integration Framework site, a BOD is created for that site for each of the templates. If you do not want these BODs to be created for you when creating sites, remove the templates. Likewise, add your own templates in *bos/template/Teamcenter* to create BODs from them for each SOA site created in the future.

Teamcenter Enterprise connector configuration elements

Teamcenter Enterprise connector configuration file

Configure the Teamcenter Enterprise connector using **the Sites page** and the procedure described in *Create and manage configuration files*. You may need to create the configuration files that inform connectors where to find the data sources to connect to multiple data sources through a single connector. Your users must have all databases that contain Teamcenter Integration Framework information in scope for the Teamcenter Enterprise user accounts that will be accessed from Teamcenter Integration Framework. Therefore, the Teamcenter Enterprise administrator or each user must set the scope appropriately in the user's database preferences.

param

PURPOSE

Specifies the parameter names and values used to configure the connector.

SYNTAX

<util:param name="*parameter-name***" value="***parameter-value***" />**

ATTRIBUTES

name

Specifies the name of a parameter. The following parameter names are used to configure a Teamcenter Enterprise connector:

Parameter name	Parameter value
DeleteMax	Specifies the maximum number of objects that a user or other client can delete using the delete API. A value of **0** indicates no limit on the number of deletions. If the criteria passed to the **delete** API result returns more than the maximum number of objects allowed, the method throws an exception, and no objects are deleted.
EMS_export_ synchronization	Specifies the synchronization mode for Data Exchange export calls. If this value is set to **true**, Data Exchange runs export calls sequentially to provide a synchronized export.
EMS_import_ synchronization	Specifies how the connector handles calls to the **importObject** method. If the value is set to true, the calls are synchronized into a single queue.
MUX_HOST	Specifies the name of a computer on which the Teamcenter Enterprise MUX is running.
MUX_PORT	Specifies the port number of a computer on which the Teamcenter Enterprise MUX is running.
THROTTLE_LIMIT	Specifies the maximum concurrent client-side connections. This limits the number of Teamcenter Enterprise server processes that Teamcenter Integration Framework consumes. This is required to prevent resource issues during Data Exchange swarming transfers using.
db2UITranslate	This parameter is for internal Teamcenter Integration Framework use only.

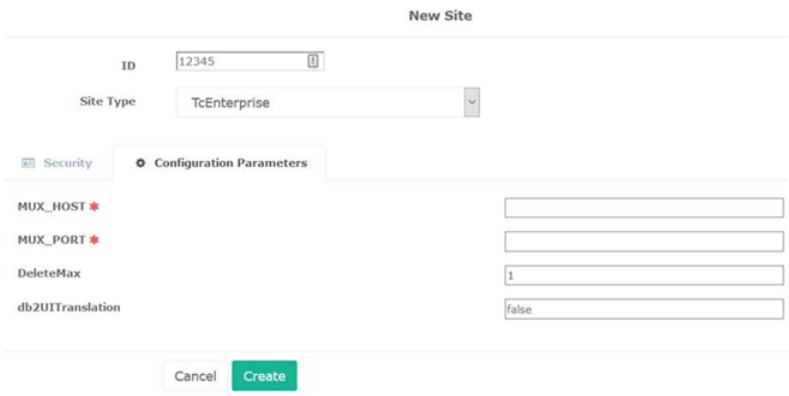
You must enter a value for this attribute.

value

Specifies the parameter value. The valid values are described in the **name** attribute description.

You must enter a value for this attribute.

EXAMPLE

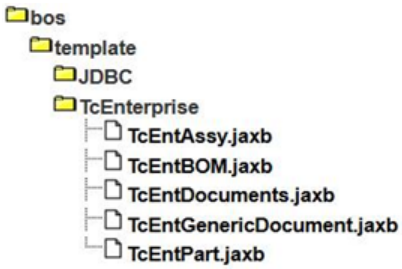


NOTES

Separate **param** elements containing **DeleteMax**, **MUX_HOST**, and **MUX_PORT** parameters are required in the Teamcenter Enterprise connector configuration file.

Teamcenter Enterprise business object definitions

Teamcenter Integration Framework provides several sample for business object definitions as follows These sample business object definitions are used in the Teamcenter Integration Framework presentation layer handlers.



An **attribute** element with a **name** attribute value of **Class** is required to upload an object to Teamcenter Enterprise. Also, an **attribute** element with a **name** attribute value of **OBID** is required to uniquely identify an object in Teamcenter Enterprise. Teamcenter Integration Framework must get and use the **OBID** value for certain functions, such as, download, checkin, and checkout. Therefore, you must include these attribute element definitions if you intend to use these functions on the business object you are defining.

Teamcenter Integration Framework business objects

Understanding business objects and business object definitions

A Teamcenter Integration Framework business object gives you a view into your data by collecting and making available logically-related pieces of data, potentially stored across multiple data sources.

Define these logical collections of information using business object definitions, or BODs, as described in **Create and manage business object definition files**. For example, you could define an employee business object as having an employee name, title, supervisor, and salary. For each of these units of information, called attributes, you also identify the data source and the data source location.

See **Business object definition configuration elements** for a description of each Teamcenter Integration Framework element you can use to create business object definitions.

Understanding user roles

You can use business object definitions to control user access to information and the actions users can perform with that information.

You control user access by defining a separate business object for each role you expect users to play. For example:

- To enable certain Teamcenter Integration Framework users to search for a corporate employee by name and view the employee's job title, supervisor, and e-mail address, you create a **Employee** business object that defines the **title**, **supervisor**, and **address** attributes.
- To enable certain users to also view the employee's salary, you create another **Employee** business object that defines the **salary** attribute in addition to **title**, **supervisor**, and **address**.

Similarly, you can control the actions different users can perform on the Teamcenter Integration Framework business object. The Teamcenter Integration Framework XML elements include elements that define user actions:

- The **delete-able** element determines whether users can remove a business object from the data source.
- The **insert-able** element determines whether users can create a business object in the data source.
- The **update-able** element determines whether a user can update an attribute in the data source.

To enable only some users to delete a business object from a data source, you must create a business object definition for each group of users.

To define a role, you create a location for each user role under the datastore **/bos** location and upload the business object definitions applicable to that role into the new role location.

After you create user roles, the security administrator assigns each user to a role location. Users can access only the business object definitions in the assigned location. You can assign role locations using XML credentials in the **BOS-box-security-info** element or you can use LDAP-based user credentials.

Configuring and managing Teamcenter Integration Framework business objects

Configuring business objects includes defining business objects for use with Teamcenter Integration Framework and mapping business object names to multiple translations for display in multiple locales.

Before you begin to define business object definitions, complete the following:

1. Determine the business objects you want to define, the information you want users to access, and the data sources that contain the information, and whether the information has different names in different data sources.

2. Determine whether to treat groups of users differently in regard to the information they can access and the actions they can perform.

3. For each data source, obtain the bean name assigned to the Teamcenter Integration Framework connector to the data source, and the configuration file name that contains information used by the connector to communicate with the data source.

4. For each unit of information you want users to access, determine the information unit name in the data source.

5. Determine relationships between business objects.

Create and manage business object definition files

1. Create a text file with the **.jaxb** extension.

   If you are using an XML editor to create the file, you may specify **business-object-definition.xsd** when the editor requests a schema. Teamcenter Integration Framework validates all business object definition files loaded into the datastore. Validating the file before uploading the file to the datastore prevents errors when you upload the file. This schema is in the datastore **bos** folder.

   > Caution:
   >
   > For Teamcenter Integration Framework to work correctly, the business object definition file must be valid according to the **business-object-definition.xsd** schema. Use a validating editor to avoid encountering errors when you upload the file to the datastore.

   Teamcenter Integration Framework provides sample business object definition files.

2. If the business object definition file contains local characters (for example, in data source or default definitions), include the encoding type in the XML header. For example:

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

3. Name the file.

   The name of this file is used as the business object name if no value is supplied for the name attribute in the XML file. For clarity, choose names that represent the business object purpose, for example: **EmployeeDetails.jaxb** and **ProjectBudget.jaxb**.

   To specify the file name, you can use from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces. Teamcenter Integration Framework does not enforce this convention, but names that do not follow the convention cause unpredictable results.

   > Note:
   >
   > You can display a different business object name to users by specifying a **label** element in the definition.

4. Define the business object using the business object definition XML elements.

5. Place the business object definition file in a temporary directory.

6. If you are implementing value mapping, open the datastore **bos** and **valuemap** folders and download the **ValueMaps.jaxb** file into a temporary directory. Modify the attribute value map.

   > Caution:
   >
   > - Do not change the value map file name or place it to another datastore location. This file must be named **ValueMaps.jaxb** and must be in the datastore **/bos/valuemap** location.

7. If you are implementing user roles, create a directory for each user role as follows:

   a. For each user role, create a temporary directory, using the role name.

   b. In each user role directory, place the business object definitions that apply to that role.

   c. Create a ZIP or JAR file containing the user role directory contents.

   d. On the Teamcenter Integration Framework Configuration Object page:

      A. Check **Check to unpack contents of jar/zip file** .

      B. Click **Browse** and navigate to the ZIP or JAR file

C. Type **/bos/***user-role-directory* in the **datastore location for object** box.

D. Click **Upload**.

8. Provide the security administrator with the names of each datastore location representing a user role.

   If your implementation does not define user roles, the datastore location for each user is the parameter value defined for the **default_user_object_dir** attribute in the business object server (BOS) connector configuration (**BOSBox.jaxb**) file.

9. If you are implementing business object definitions to run in multiple locales, create the ID map.

> Note:
>
> To revise a business object definition, download the file from the datastore **bos** folder, edit the file, and upload the edited file to the **/bos** location. If you are not using a validating XML editor, to avoid errors, validate against the **business-object-definition.xsd** schema before you upload the file.

Building a business object definition

Defining the basic structure

The XML elements shown in the following example provide the basic definition of a business object and are required for all business object definitions in the structure shown:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--
bcprt
This software and related documentation are proprietary to Siemens Digital
Industries Software. COPYRIGHT 2005 UGS CORP.  ALL RIGHTS RESERVED
ecprt
-->
<business-object-definition name="Employee"
                            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                            xmlns:conn="http://teamcenter.com/integrator/connection/
2005-06"
                            xsi:noNamespaceSchemaLocation="business-object-
definition.xsd"
                            xmlns="http://teamcenter.com/integrator/bod/2005-06">
  <label ID="Employee.name"/>
  <data-source name="Local DB2">
    <data-segment name="A">
      <conn:data-source-spec bean-name="JDBCBox" config-name="Db2DatabaseConfig"/>
      <class-name>employee</class-name>
      <insert-able>true</insert-able>
      <delete-able>true</delete-able>
      <compute-able>true</compute-able>
      <attribute name="EMPNO">
        <label default="Employee No." ID="Employee.attr.EMPNO"/>
        <name-on-source>EMPNO</name-on-source>
```

```
        <primary-key>true</primary-key>
        <search-able>true</search-able>
        <link name="All Employees" href="search?Object=Employee&Fields=0">
           <label ID="SAMPLE.link.AllEmp"/>
        </link>
        <link name="All Departments" href="search?Object=Department&Fields=0">
           <label ID="SAMPLE.link.AllDept"/>
        </link>
        <link name="Department Manager" href="expand?Object=Department&Relation=mgr&
Fields=1&FieldName1=DEPTNO&FieldValue1=this.WorkDept">
           <label ID="Employee.link.DeptMgr"/>
        </link>
      </attribute>
      <attribute name="FirstName">
        <label default="First Name" ID="Employee.attr.FirstName"/>
        <name-on-source>FirstNme</name-on-source>
        <required-for-insert>true</required-for-insert>
        <search-able>true</search-able>
        <update-able>true</update-able>
      </attribute>
      <attribute name="LastName">
        <label default="Last Name" ID="Employee.attr.LastName"/>
        <name-on-source>LastName</name-on-source>
        <search-able>true</search-able>
        <update-able>true</update-able>
      </attribute>
      <attribute name="MI">
        <label default="Initial" ID="Employee.attr.MI"/>
        <name-on-source>MidInit</name-on-source>
        <update-able>true</update-able>
    <valid-value>A</valid-value><valid-value>B</valid-value><valid-value>C</valid-value>
 <valid-value>D</valid-value><valid-value>E</valid-value><valid-value>F</valid-value>
 <valid-value>G</valid-value><valid-value>H</valid-value><valid-value>I</valid-value>
 <valid-value>J</valid-value><valid-value>K</valid-value><valid-value>L</valid-value>
    <valid-value>M</valid-value><valid-value>N</valid-value><valid-value>O</valid-value>
 <valid-value>P</valid-value><valid-value>Q</valid-value><valid-value>R</valid-value>
 <valid-value>S</valid-value><valid-value>T</valid-value><valid-value>U</valid-value>
 <valid-value>V</valid-value><valid-value>W</valid-value><valid-value>X</valid-value>
    <valid-value>Y</valid-value><valid-value>Z</valid-value>
      </attribute>
      <attribute name="WorkDept">
        <label default="Department" ID="Employee.attr.WorkDept"/>
        <name-on-source>WorkDept</name-on-source>
        <update-able>true</update-able>
        <dynamic-valid-value object="Department" attribute="DEPTNO"/>
        <link href="search?Object=Department&Fields=1&FieldName1=DEPTNO&FieldVal
ue1=this.WorkDept&isValueLiteral1=true"></link>
      </attribute>
      <attribute name="PhoneNo">
        <label default="Phone No." ID="Employee.attr.PhoneNo"/>
        <name-on-source>PhoneNo</name-on-source>
        <update-able>true</update-able>
      </attribute>
      <attribute name="HireDate">
        <label default="Hire Date" ID="Employee.attr.HireDate"/>
        <name-on-source>HireDate</name-on-source>
        <update-able>true</update-able>
      </attribute>
      <attribute name="Title">
        <label default="Title" ID="Employee.attr.Title"/>
```

```
        <name-on-source>Job</name-on-source>
        <search-able>true</search-able>
        <value-map-name>EmployeeTitleMap</value-map-name>
        <update-able>true</update-able>
      </attribute>
      <attribute name="EdLevel">
        <label default="Educ. Level" ID="Employee.attr.EdLevel"/>
        <type>int</type>
        <search-able>true</search-able>
        <hidden>true</hidden>
      </attribute>
      <attribute name="Sex">
        <label default="Sex" ID="Employee.attr.Sex"/>
        <hidden>true</hidden>
      </attribute>
      <attribute name="BirthDate">
        <label default="Birthday" ID="Employee.attr.BirthDate"/>
        <hidden>true</hidden>
      </attribute>
      <attribute name="Salary">
        <label default="Salary" ID="Employee.attr.Salary"/>
        <type>double</type>
        <update-able>true</update-able>
      </attribute>
      <attribute name="Bonus">
        <label default="Bonus" ID="Employee.attr.Bonus"/>
        <type>double</type>
        <update-able>true</update-able>
      </attribute>
      <attribute name="Comm">
        <label default="Commission" ID="Employee.attr.Comm"/>
        <type>double</type>
        <update-able>true</update-able>
      </attribute>
    </data-segment>
  </data-source>
</business-object-definition>
```

business-object-definition

Defines a Teamcenter Integration Framework business object. The XML elements specified between the **business-object-definition** element beginning and ending tags compose the definition of one business object. The name is optional; Teamcenter Integration Framework uses the XML file name defining the business object.

data-source

Defines a conceptual name for the data source containing the business object information. Provide any name that helps describe the data source. For example, if the data source is in a DB2 database, you might define a conceptual data source named **Local DB2**. The **data-source** element does not define the data source containing the information. That is defined in the **conn:data-source-spec** element.

One **data-source** element is required. You can have multiple **data-source** elements, and each name must be unique within the business object definition.

data-segment

Defines the business object information stored in one data source (**data-source-spec** element).

The XML elements specified between the beginning and ending **data-segment** tags identify the data source containing the information, the object name, database table, or other entity in the data source, and each unit of information you want to include.

data-source-spec

Specifies the data source that stores the information. You provide the bean name of a Teamcenter Integration Framework connector and the configuration file name that provides information for communicating with the data source. This element in defined in the **conn** namespace, therefore it must be preceded by this namespace designation (**data-source-spec**).

The preceding example defines the JDBC connector for the DB2 data source.

class-name

Specifies the information entity name in the data source. This is a class name in an object-oriented system, a table name in a database-oriented data source, and a file name in a file-system data source.

attribute

Defines a unit of information maintained in the data source specified with the **data-source-spec** element. When a data source stores multiple units of information for the business object, there are multiple **attribute** elements.

You can define an attribute in the following ways:

- As the value returned from a data source.

  You define this type of attribute using only XML elements.

- As a dynamic value based on the values of other attributes.

  You define this type of attribute using XML elements and a Tool Command Language (Tcl) script. The Tcl script determines the attribute value by performing operations on the values of other attributes. See also **dynamic-attribute**, **script-text**, and **script-file**. The **EmployeeScript.jaxb** file is a sample business object definition using Tcl scripts.

> Note:
>
> These XML elements are the basic required elements for defining a business object whose information is stored in only one data source. To define business objects whose information is stored in multiple data sources, you must define them as aggregate or multisource business objects.

Defining aggregate business objects

A **data-segment** element defines all attributes and properties associated with the business object in a single data source (**data-source-spec** element). When some attributes of your business object are stored

on one data source, and other attributes are stored on another data source, you define the business object using multiple **data-segment** elements.

For example, if the part number and nomenclature attributes for the **PartInventory** business object reside in a Teamcenter Enterprise data source, but its inventory quantity and cost reside on a JDBC data source, you specify two data segments within the **data-source-spec** element, one for each connector.

The **data-source-spec** element provides a conceptual name that helps describe and group the data sources. For example, if all the data sources are located in Chicago, you could use **Chicago** as the **data-source-spec** element name.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<business–object–definition name="PartInventory">
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:conn="http://teamcenter.com/integrator/connection/
2005-06"
                  xsi:noNamespaceSchemaLocation="business-object-
definition.xsd"
                  xmlns="http://teamcenter.com/integrator/bod/2005-06">
   <data–source name="Chicago">
      <data–segment name="Part">
         <conn:data-source-spec bean-name="TcEntBox"
               config-name="TcEntConfig"/>
         <class–name>Part</class–name>
         <attribute name="PartNumber">
               :
         </attribute>
         <attribute name="Nomenclature">
               :
         </attribute>
      </data–segment>
      <data–segment name="Inventory">
         <multiple-results>true</multiple-results>
         <conn:data-source-spec bean-name="JDBCBox"
               config-name="Inventory”/>
         <class–name>Inventory</class–name>
    <util:where-clause>PartNumber = Part.PartNumber</util:where-clause>
         <attribute name="Quantity">
               :
         </attribute>
         <attribute name="PartCost">
               :
         </attribute>
      </data–segment>
   </data–source>
</business–object–definition>
```

Teamcenter Integration Framework searches for each data segment serially, in the order you specify the data segments. The resulting business object is an *aggregate* of attributes from different data sources.

The first data segment you list in the data definition is the *primary data segment*. All succeeding data segments are *secondary data segments*. Secondary data segments require an additional XML element, **where-clause** (aggregate) or **expand**. The **where-clause** element defines the attribute values of a previously defined data segment that are sent to the secondary data source to determine the instance used for data aggregation. The **expand** element finds attributes by expanding a specified relation formally modeled in the secondary data source. In the standard connectors supplied with Teamcenter Integration Framework, you can use the **expand** element only with the connector to Teamcenter Enterprise. Your enterprise can create a custom connector that supports this element.

Defining multisource business objects

If an attribute that your business object requires might be stored in multiple data sources, define the business object using multiple **data-source** elements.

For example, if the nomenclature attribute for the **Part** business object resides in either a Teamcenter Enterprise data source or a JDBC data source, define a **data-source** element for each data source that contains the information:

- Within each **data-source** element, you must define identical attributes.
- Within each **data-source** element, you can define one or more **data-segment** elements. The **data-segment** elements described a related collection of data within the data source.

You can use the **data-source** name to provide additional information, for example, the data source locations.

Teamcenter Integration Framework searches the defined data sources in parallel to retrieve the data:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<business-object-definition name="Part">
   <data-source name="Chicago">
            <data-segment name="Part1">
                  <source-name>TcEntBox</source-name>
                  <class-name>Part</class-name>
                  <attribute name="PartNumber">
                        :
                  </attribute>
                  <attribute name="Nomenclature">
                        :
                  </attribute>
            </data-segment>
   </data-source>
   <data-source name="Tokyo">
            <data-segment name="Part2">
                  <source-name>JDBCBox:PartDB</source-name>
```

```
                    <class-name>PartTable</class-name>
                    <attribute name="PartNumber">
                          :
                    </attribute>
                    <attribute name="Nomenclature">
                          :
                    </attribute>
               </data-segment>
   </data-source>
</business-object-definition>
```

Defining user actions

For each business object, you can define the actions that users can perform. Each of these elements is optional. If you do not specify the element, the client cannot perform the associated action:

search-able

For users to search data sources for business object attributes by name, the attribute must exist in the first data source Teamcenter Integration Framework searches. For this reason, you must define those attributes in the primary data segment (see *Defining aggregate business objects*) and specify the **search-able** element within the **attribute** element.

update-able

To allow a user to update a business object attribute in a data source, define the **update-able** element within the **attribute** element.

If you define an attribute as required to determine instance uniqueness (using the **primary-key** element), the user cannot update the attribute. Do not specify both a **primary-key** and an **update-able** element for the same attribute.

insert-able

To allow users to create a business object instance in the data source, specify the **insert-able** element within the **data-segment** element.

delete-able

To allow users to remove a business object instance in the data source, specify the **delete-able** element within the **data-segment** element.

default-value

You can define default values Teamcenter Integration Framework provides when inserting (creating) a business object by specifying the **default-value** element within the **attribute** element. Teamcenter Integration Framework displays the default value to the user in the form's attribute box.

valid-value

You can specify valid values for an attribute by specifying the **valid-value** element within the **attribute** element. Teamcenter Integration Framework displays the valid values as choices in the form's attribute box.

Defining value maps for attributes

Two data sources often store different values for the same attribute. For example, in one data source the values of a part attribute may be stored as **Make** and **Buy**, and the values for same attribute in another data source may be stored as **1** and **2**. To transfer data between the two data sources with the correct values for each, you can map the values for these attributes so that Teamcenter Integration Framework converts them to the appropriate value when data is transferred. The mapping between data sources is accomplished by mapping the data source values to a common Teamcenter Integration Framework business object value.

For each business object attribute you want to map values for, create a value map unique to the data source. For example, for the **MakeBuyIndicator** attribute in a Teamcenter Integration Framework business object defined for a PDM data source, create a value map named **MakeBuyIndicatorPDM**; for the same attribute in an ERP data source, create a value map named **MakeBuyIndicatorERP**:

```
<?xml version="1.0" encoding="UTF-8"?>
<value-map-config>
   <value-map name="MakeBuyIndicatorPDM">
      <mapping BOS-value="AccMake" source-value="Make"/>
      <mapping BOS-value="AccBuy" source-value="Buy"/>
   </value-map>
   <value-map name="MakeBuyIndicatorERP">
      <mapping BOS-value="AccMake" source-value="1"/>
      <mapping BOS-value="AccBuy" source-value="2"/>
   </value-map>
</value-map-config>
```

Within each value map, use the **mapping** element to define the value used by the Teamcenter Integration Framework business object server (**BOS-value**) and map it to the corresponding value for the data source (**source-value**). To exchange data between data sources, the **BOS-value** must match in each value map. For example, to exchange data between the PDM data source and the ERP data source in the above example, the **BOS-value** attributes for each map are **AccMake** and **AccBuy**.

After creating the value map, enter its name in the attribute definition for the Teamcenter Integration Framework business object definition using the **value-map-name** element. For example, for the attribute defined in the PDM business object definition, enter:

```
<attribute name="MakeBuyIndicator">
   <value-map-name>MakeBuyIndicatorPDM</value–map–name>
   <update-able>true</update-able>
</attribute>
```

For the attribute defined in the ERP business object definition, enter:

```
<attribute name="MakeBuyIndicator">
   <value–map–name>MakeBuyIndicatorERP</value–map–name>
   <update-able>true</update-able>
</attribute>
```

If business object attributes have associated value maps, database searches using these attributes also have value mapping applied. For example, if the **Employee** business object definition has a **Title** attribute with an associated **EmployeeTitleMap** value map that maps the business object value **President** to the data source value **PRES**, **President** should be used in searches (user interface and reactors).

In addition, when processing **where-clause XML** elements in secondary data segments, Teamcenter Integration Framework processes values of mapped attributes in terms of business object values. For example, if a **where-clause** element specifies **B_Att = A.A_Att** and **A_Att** has an associated value map, the mapped business object value is used to compare to the value of **B_Att**. In this case, **B_Att** and **A_Att** should use the same value map to ensure that the business object value obtained using the **A.A_Att** reference is mapped to the source value using the value map associated with **B_Att**.

The following conditions apply to value maps:

- Mapping values is case sensitive in both directions (between business object server and data source) unless you include the **ignore-case** element with the value set to **true**. Using case-sensitive value maps can cause unexpected behavior when database engines ignore case for searches. For example, if the **MakeBuyIndicator** attribute uses a value map that maps **Make** on **M**, and the case-insensitive database stores the attribute as **m**, a search on **Make** succeeds (using **M**), but the returned value **m** is not mapped. Therefore, for databases that ignore case, include the **ignore-case** element.

- You cannot use a value map for dynamic attributes, that is, attributes defined in terms of other attributes using a Tcl script.

- You cannot use value mapping for localization. Teamcenter Integration Framework does not support the translation of data that is stored in databases.

- Attribute value mapping does not support type conversions. It does not convert strings to integers. If the mapped attribute is a string, the mapped value is a string.

- You cannot use a value map for valid value elements. The attribute values of valid value elements are stated as business object values; they are not specific to a data source. Values used for a query are also stated in terms of business object values.

- If you expand or query from business object A to business object B, where the primary key in business object A used for finding business object B is value-mapped, the corresponding key in business object B must also be value-mapped. For example, assume business object A has an ID with value map **acc01 01** when expanded to B. If B's primary key is **Bld** with no value map, the constructed **where-clause** element used to find B is:

  ```
  where Bld = 'acc01'
  ```

  By applying the same value map to **Bld**, the correct value is used to find B.

- Teamcenter Integration Framework does not support the use of value mapping with the **compute** and **execute** methods.

Defining relationships and links

To enable clients to find business objects by expanding relationships, you can define a relationship within a data segment. Because the data segment defines a single data source, you can use the relationship name as it is modeled by that data source. For example, if the data source is a product knowledge management (PKM) system, you can create a relation between business objects using the name of a relation class in the PKM system.

When the data source does not model relationships, or if you want to create a relationship for which a data source does not have a class, you can create the relationship using a **where-clause** (relationship) element (foreign key join).

relation

Defines a Teamcenter Integration Framework relationship and specifies the name by which the Teamcenter Integration Framework business object server references the relationship.

result-name

Defines the type of business objects returned when the relationship is expanded.

formal-name or where-clause (relationship)

For a data source that formally models relationships between business objects, use the **formal-name** element to specify the relationship to be expanded. For a data source that does not model relationships, use the **where-clause** (relationship) element to specify the relationship result object instances.

link

You can relate a business object to another business object or to URL references. The Teamcenter Integration Framework user interface displays named links in a shortcut menu.

If you are defining an aggregate business object, you can use the **expand** element in secondary data segments instead of **where-clause** (aggregate). The **expand** element obtains attributes from a data source by expanding a relationship.

Tailoring business object names for display

The convention for Teamcenter Integration Framework names allows you to specify from 1 to 24 ASCII alphanumeric characters and underscores, beginning the name with an alphabetical character and excluding embedded blank spaces. These names are used internally by Teamcenter Integration Framework and, by default, displayed to the user. (Data source names, specified with the **name** attribute, allow a different set of characters.)

If you want to display a Teamcenter Integration Framework name that does not conform to these conventions, include a **label** element in the definition. The **label** element allows you to specify a different business object name (on the **business-object-definition** element) or attribute name (on the **attribute** element) using any characters or character sets. You can also begin a name with a number and include blank spaces.

For the following definition, Teamcenter Integration Framework displays the **PartInventory** business object to the user as **Part Inventory**:

```
<business-object-definition name="PartInventory">
        <label default="Part Inventory"/>
                :
</business-object-definition>
```

For the following example, Teamcenter Integration Framework displays the **ThreeDimenImage** attribute as **3D Image**:

```
<attribute name="ThreeDimenImage">
        <label default="3D Image"/>
                :
</attribute>
```

The preceding label definitions are sufficient for providing one tailored or localized display name for a business object or attribute. You can also **map multiple localized names** for a business object. At runtime, Teamcenter Integration Framework chooses a localized name for the business object based on the user's locale.

Defining presentation elements

Within a business object definition, you can define two elements for presenting a business object in the presentation layer, **icon-name** and **default-color**.

This minimal control of presentation is intended for testing and for creating prototype demonstrations. The primary control of Teamcenter Integration Framework presentation is the Extensible Stylesheet Language (XSL) style sheet included with the Teamcenter Integration Framework user interface. The Teamcenter Integration Framework XSL defines the presentation for the XML elements that define business objects. The Teamcenter Integration Framework XSL overrides the business object definition XML elements. You can customize the user interface elements.

You can define the default icon that Teamcenter Integration Framework displays for a business object using the **icon-name** element and the color for the data segment using the **default-color** element.

Concatenating text

Within Teamcenter Integration Framework business object definitions, you can combine multiple pieces of text data into a single text string by concatenating text as follows:

'*text*' **+** '*text*' ...

Use this technique when creating a single comparison entity from multiple attributes in a **where-clause** element or condition.

The following example combines a part number, revision, and sequence number from a primary data segment into a single string reference for the secondary **where-clause** element. Bold text highlights the text concatenation.

```
<data-segment name="Part">
      <source-name>JDBCBox:Employee</source-name>
      <class-name>Part</class-name>
      <attribute name="PartNo">
         <primary-key>true</primary-key>
         <search-able>true</search-able>
      </attribute>
      <attribute name="Revision">
                :
       </attribute>
      <attribute> name="SequenceNo">
                :
       </attribute>
</data-segment>
<data-segment name="Inventory">
      <source-name<JDBCBox:Inventory</source-name>
      <class-name>"Inventory</class-name>
      <util:where-clause>PartNumber = 'Part.PartNo' + 'Part.Revision' +
                    'Part.SequenceNo'</util:where-clause>
      <attribute name="PartNumber">
                :
       </attribute>
      <attribute name="Revision">
                :
       </attribute>
      <attribute> name="SequenceNum">
                :
       </attribute>
      <attribute name="Quantity">
                :
       </attribute>
      <attribute> name="Cost">
                :
       </attribute>
</data-segment>
```

Substituting text

Business object definitions text substitution methods

Within Teamcenter Integration Framework business object definitions, you can use several types of text substitution:

- *XML entity substitution* is a static substitution method that you can define once and use anywhere.
- *Local data segment attribute substitution* allows you to substitute run-time values of attributes into **link** element values when the attribute and **link** elements are defined within the same data segment.

- *External data segment attribute substitution* allows you to substitute run-time values of attributes into secondary **where–clause** element when the attributes and the clauses are defined in different data segments.

Substituting local attribute values

To insert the run-time values of attributes into **where-clause** elements, conditions, and URL references within the data segment the attribute is defined in, use the following format:

this.*attribute-name*

In the following example, the data segment includes three attribute definitions, **EMPNO**, **LastName**, and **FirstName**. To substitute run-time values for these attributes in the sample user interface shortcut menu, the example specifies **this.EMPNO**, **this.FirstName**, and **this.LastName** references. The attributes are defined in the same data segment as the references to them.

```
<data–segment name="A">
      <source–name>JDBCBox:Employee</source–name>
      <class–name>employee</class–name>
      <attribute name="EMPNO">
         <primary-key>true</primary-key>
         <search–able>true</search–able>
         <link name="Inventory"
             href="search?Object=Inventory&
                   Fields=1&FieldName1=RespPerson&
                   FieldValue1=this.EMPNO&/>"
         <link name="DeptInfo"
             href="search?Object=DeptInfo&
                   Fields=2&FieldName1=FName&
                   FieldValue1=this.FirstName&
                   FieldName2=LName&
                   FieldValue2=this.LastName&/>"
      </attribute>
      <attribute name="LastName">
                :
      </attribute>
      <attribute> name="FirstName">
                :
      </attribute>
</data–segment>
```

Substituting external attribute values

To insert the run-time value of an attribute defined in a different data segment, use the following format:

*segment-name***.***attribute-name*

In the following example, the **Part** data segment defines the **Part** class and three attributes, **PartNo**, **Revision**, and **SequenceNo**. The **Inventory** data segment refers to these attributes by class name and attribute name: **Part.PartNo**, **Part.Revision**, and **Part.SequenceNo**. At runtime, Teamcenter Integration Framework substitutes the attribute values for these references.

> Note:
>
> You can substitute external attribute values only in the secondary data segments of aggregate business objects. Because each data segment is evaluated in the order it is defined in the business object, any attribute referenced must be part of a previously defined data segment.

```
<business-object-definition name="Part">
      <source-name>JDBCBox:Employee</source-name>
      <class-name>Part</class-name>
      <attribute name="PartNo">
         <primary-key>true</primary-key>
         <search-able<true</search-able>
      </attribute>
      <attribute name="Revision">
                :
      </attribute>
      <attribute> name="SequenceNo">
                :
      </attribute>
</business-object-definition>
<business-object-definition name="Inventory">
      <source-name>JDBCBox:Inventory</source-name>
      <class-name>Inventory</class-name>
      <where-clause>PartNumber = 'Part.PartNo' and
                    Revision = 'Part.Revision' and
                    SequenceNum = 'Part.SequenceNo'</where-clause>
      <attribute name="PartNumber">
                :
       </attribute>
      <attribute name="Revision"/>
                :
       </attribute>
      <attribute> name="SequenceNum"/>
                :
       </attribute>
      <attribute name="Quantity"/>
                :
       </attribute>
      <attribute> name="Cost"/>
                :
       </attribute>
</business-object-definition>
```

Modify the attribute value map file

Teamcenter Integration Framework provides an attribute value map file in the datastore **bos→valuemap** folder. You can add value maps to this file.

> Caution:
>
> Do not change the value map file name or place it to another datastore location. This file must be named **ValueMaps.jaxb** and must be in the datastore **/bos/valuemap** location.

1. Download the **ValueMaps.jaxb** and **value-map.xsd** files into a temporary directory.
2. If you are using an XML editor to modify the file, specify **value-map.xsd** when the editor requests a schema reference.

   > Caution:
   >
   > For Teamcenter Integration Framework to work correctly, the Teamcenter Integration Framework message server configuration file must be valid according to the **value-map.xsd** schema. Use a validating editor to avoid encountering errors after you upload the file to the datastore.

3. Define the **value mapping using XML elements**.
4. If the attribute value map file contains local characters, include the encoding type in the XML header. For example:

   ```
   <?xml version="1.0" encoding ="euc_jp"?>
   ```

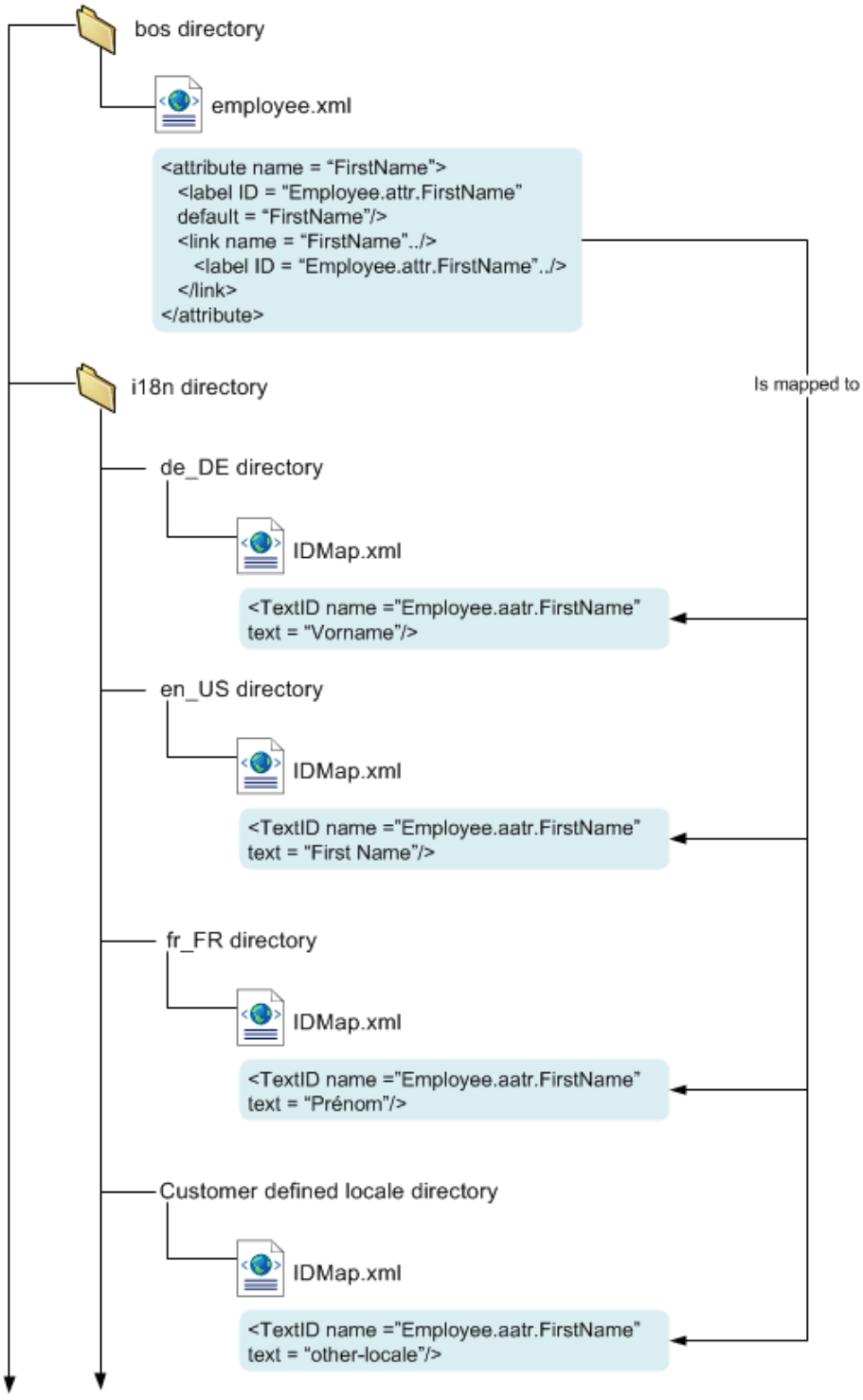
5. Upload the **ValueMaps.xml** file to the datastore **/bos/valuemap** location.

Localizing business object definitions

Mapping business object names to multiple locales

Siemens Digital Industries Software provides Teamcenter Integration Framework for the multiple locales. You can provide additional locales within ID maps.

To localize your business object definitions for use in multiple locales, you assign an ID to the name in the business object definition and map that ID to localized text in an *ID map*, one for each locale as shown in the following figure.



ID mapping to multiple locales

Teamcenter Integration Framework checks the user's locale, finds the corresponding locale in the **i18n** datastore folder, finds the ID for the name in **IDMap.xml** file, and displays the localized text.

To create the ID map, you use Teamcenter Integration Framework-provided XML elements. Teamcenter Integration Framework also provides a document type definition (DTD) file that defines the XML elements and structure for ID maps. The DTD file is named **IDMap.dtd** and is in the **i18n→common** datastore folder. This one DTD applies to all locales.

Create ID maps

1. In the business object definition, define an **ID** attribute on either the **label** element (for a business object name or attribute name) or the **link** element (for a named link).

   For example, in the **Employee** business object definition, you could define the following IDs:

```
<attribute name="EMPNO">
   <label default="Employee No." ID="Employee.attr.EMPNO"/>
   <name-on-source>EMPNO</name-on-source>
   <primary-key>true</primary-key>
   <search-able>true</search-able>
      <link name="All Employees" href="search?Object=Employee&Fields=0">
         <label ID="SAMPLE.link.AllEmp"/>
      </link>
      <link name="All Departments" href="search?Object=Department&Fields=0">
         <label ID="SAMPLE.link.AllDept"/>
      </link>
      <link name="Department Manager"
href="expand?Object=Department&Relation=mgr&Fields=1&FieldName1=DEPT
NO&FieldValue1=this.WorkDept">
         <label ID="Employee.link.DeptMgr"/>
      </link>
</attribute>
```

   When used with **ID**, the **default** attribute is optional; it specifies the text Teamcenter Integration Framework displays if it cannot find a localized name.

2. For each locale in the Teamcenter Integration Framework environment, create a file named **IDMap.xml**.

   - If you are using an XML editor to create the file, specify **IDMap** when the editor requests a DTD.

     > Note:
     >
     > Siemens Digital Industries Software recommends using a validating editor to avoid errors due to invalid XML files.
     >
     > If do not have the **IDMap.dtd** file available, download the file into a temporary directory from the Teamcenter Integration Framework datastore.

   - Include a **DOCTYPE** element in the file with a reference to **IDMap.dtd**:

```
<!DOCTYPE IDMap SYSTEM "..\common\IDMap.dtd">
```

3. Using **ID map XML elements**, map the ID in the business object definition to a localized name. For example:

   In the ID map for the German locale:

   ```
   <TextID name="Employee.attr.FirstName" text="Vorname"/>
   ```

   In the ID map for the French locale:

   ```
   <TextID name="Employee.attr.FirstName" text="Prénom"/>
   ```

4. Upload each ID map file into the appropriate locale under the datastore **/i18n** location.

5. The **idmap.cache.ttl** property in the **globalservices.properties** file sets the time period that Teamcenter Integration Framework uses to automatically reload the ID map file. The file is reloaded after the period specified for this property expires. If you want to reload the file immediately, restart Teamcenter Integration Framework.

Business object definition configuration elements

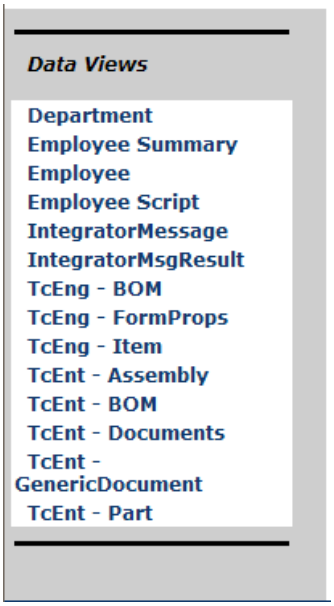
Sample business object definitions

Teamcenter Integration Framework provides several sample business object definition files. These sample business object definitions are displayed on the Teamcenter Integration Framework portal web page in the **Data Views** list and are used in the Teamcenter Integration Framework presentation layer.

Following are some of the business object definitions Teamcenter Integration Framework provides:

Message.jaxb
MsgResult.jaxb
Department.jaxb
EmpDept.jaxb
Employee.jaxb
EmployeeScript.jaxb

The sample business object definitions are in the **initial_datastore_bos_samples.zip** file and are also loaded into the **BOS** datastore location.



Data views

attribute

PURPOSE

Defines the basic unit of information associated with a business object. You define an attribute as the value returned from a data source.

SYNTAX

<attribute name="*attribute-name***">**
<util:param ... />
:
<util:param ... />
<label ... />
<name-on-source>...</name-on-source>
<type> ... </type>
<size>... </size>
<primary-key> ... </primary-key>
<required-for-insert> ... </required-for-insert>
<default-value> ... </default-value>
<search-able> ... </search-able>
<value-map-name> ... </value-map-name>
<sort-key> ... </sort-key>
<update-able> ... </update-able> or **<hidden> ... </hidden>**
<valid-value> ... </valid-value> or **<dynamic-valid-value ... />**
:
<valid-value> ... </valid-value>
<link ... > ... </link>
:
<link > ... </link>
:
</attribute>

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the elements in the order shown in the **attribute** element syntax.

ATTRIBUTES

name

Specifies the business object attribute name.

Business object server requests reference the attribute by this name, which can be different from the name that the attribute is known by within the data source.

This name is required. The name can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces.

You can change this name for display to Teamcenter Integration Framework users with the **label element**.

EXAMPLE

```
<attribute name="PartNumber">
      :
</attribute>
<attribute name="Revision">
      :
</attribute>
```

NOTES

- An **attribute** or **dynamic-attribute** element is required for a data segment.

  A data segment can contain any number of **attribute** elements.

- If the data source requires that a user enter a value for an attribute when creating a business object instance, you must specify the attribute within the business object definition.

- You can include any number of **valid-value** elements or a single **dynamic-valid-value** element.

- See also *name-on-source*, *default-value*, *label*, *link*, *param*, *primary-key*, *search-able*, *type*, *update-able*, *hidden*, *valid-value*, *dynamic-valid-value*, and *value-map-name*.

business-object-definition

PURPOSE

Defines the business object definition's root element.

SYNTAX

<business-object-definition name="*business-obj-name***">**
<util:param ... />
<label ... />
<sort-direction> ... </sort-direction>
<icon-name> ... </icon-name>
<data-source ... >
:
</data-source>
:
<data-_source ... >
:
</data-source>
</business-object-definition>

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the business object definition elements in the order shown in the element syntax.

ATTRIBUTES

name

Specifies the business object name.

The **name** attribute is optional. If you do not specify this attribute, Teamcenter Integration Framework uses the XML file's base name. If you specify this attribute, ensure that the name matches the XML file's base name.

The name can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces. You can change this name for display to Teamcenter Integration Framework users with the **label** element.

EXAMPLE

```
<business-object-definition name="Part">
      :
      :
</business-object-definition>
```

NOTES

- The **business-object-definition** element is required for each business object definition file.

  A business object definition file can have only one **business-object-definition** element.

- All other business object definition elements are contained within the data definition.
- See also *data-source*, *icon-name*, *label*, *param*, and *sort-direction*.

class-name

PURPOSE

Specifies the name of a data source entity that contains the data associated with the business object. For example, a class name in an object-oriented data source, a table name in a database-oriented data source, and a file name in file-system data source.

SYNTAX

<class-name>*entity-name***</class-name>**

ATTRIBUTES

None.

EXAMPLE

```
<class-name>Part</class-name>
<class-name>EmpTable</class-name>
```

NOTES

- The **class** element is required for a data segment.

  A data segment can have only one **class** element.

- See also *data-segment*.

compute-able

PURPOSE

Indicates whether the connector associated with a data segment supports the compute functions (**AVG**, **COUNT**, **MIN**, **MAX**, and **SUM**). If set to **true**, Teamcenter Integration Framework users can apply compute functions to the attributes in the data segment.

SYNTAX

<compute-able>*true-or-false***</compute-able>**

ATTRIBUTES

None.

EXAMPLE

```
<compute-able>true</compute-able>
```

NOTES

- The **compute-able** element is optional.

  If you do not specify a **compute-able** value, Teamcenter Integration Framework uses **false**.

  A data segment can have only one **compute-able** element.

- A user can apply a compute function to any attribute in the data segment. However, only numeric attributes can support the **AVG** and **SUM** functions.

- See also *data-segment*.

data-segment

PURPOSE

Defines a basic unit of data encapsulation within a business object. The **data-segment** element defines all the attributes and properties associated with the business object in a single data source.

SYNTAX

<data-segment name="*data-segment-name***">**
<util:param name="merge-duplicate-cells" value="true" or **"false" />**
<util:param ... />
:
<util:param ... />
<multiple-results> ... </multiple-results>
<conn:data-source-spec ... />
<class-name> ... </class-name>
<default-color> ... </default-color>
<insert-able> ... </insert-able>
<delete–able> ... </delete–able>
<compute-able> ... </compute-able>
<relation ... >
:
</relation>
:
<relation ... >
:
</relation>
<expand> ... </expand> or **<util:where-clause> ... </util:where-clause>**
<attribute ... > or **<dynamic-attribute ... >**
: **:**
</attribute> **</dynamic-attribute>**
:
<attribute ... > or **<dynamic-attribute ... >**
: **:**
</attribute> **</dynamic-attribute>**
</data-segment>

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the elements in the order shown in the **data-segment** element syntax.

ATTRIBUTES

name

Specifies the data segment name.

The name can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces.

You use the data segment name when referencing attribute values within data segment definitions.

EXAMPLE

```
<data-segment name="prime">
      :
      :
</data-segment>
```

NOTES

- One **data-segment** element is required for a **data-source** definition.

  A **data-source** definition can have any number of **data-segment** elements.

- Multiple data segments indicate an *aggregate business object* that combines information from several data sources.

- The first data segment defined in a data source is the *primary data segment*. The data source defined for the primary data segment contains the main information for the business object. All other data segments defined in a data source are secondary data segments. When a Teamcenter Integration Framework user queries for a business object, the business object server searches the primary data segment first. The information, returned in a data set after querying the primary data source, is used in subsequent requests to data sources defined by other data segments.

- If you define more than one **data-source** element within a business object definition (a multisource object), the data segments within each **data-source** element must contain identical definitions for the same attributes. Other elements within the data segment can vary; in particular, **conn:data-source-spec**can define a different data source for each attribute and **class-name** a different name within the data source.

- For each **data-segment** element, you can have multiple **attribute** or **dynamic-attribute** elements. However, you cannot mix the two element types.

- For each **data-segment** element, you can have one **param** element with the **merge-duplicate-cells** name.

- See also *attribute*, *class-name*, *compute-able*, *data-source*, *default-color*, *delete-able*, *insert-able*, *multiple-results*, *param*, *data-source-spec*, *relation*, *expand*, *dynamic-attribute*, and *where-clause (aggregate)*.

data-source

PURPOSE

Defines a conceptual name for the data source containing the business object information. More than one **data-source** element for a business object definition indicates to Teamcenter Integration Framework that the business object attributes may be stored in more than one data source.

SYNTAX

<data-source name="*data-source-name***">**
<util:param ... />
:
<util:param ... />
<data-segment ... >
:
</data-segment>
<data_segment ... >
:
</data-segment>
</data-source>

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the elements in the order shown in the **data-source** element syntax.

ATTRIBUTES

name

Specifies the data source name.

The name can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces.

Each data source must have a unique name within the business object.

EXAMPLE

```
<data-source name="Chicago">
      :
</data-source>
```

NOTES

- One **data-source** element is required for a data definition.

A data definition can have any number of **data-source** elements.

- When you define multiple data sources for a data definition, you must define the same attributes for each **data-source** group.
- When multiple **data-source** elements are defined for a data definition, the business object server searches for the attributes in each defined data source concurrently.
- See also *data-segment* and *param*.

data-source-spec

PURPOSE

Defines the source for the attribute data defined within a data segment. This element is defined in the **conn** namespace.

SYNTAX

<conn:data-source-spec bean-name="*connector-bean-name***" config-name="***connector-configuration-name***"/>**

ATTRIBUTES

connector-bean-name

Specifies the Java naming and directory interface (bean) name of a Teamcenter Integration Framework connector.

The **bean name** of a Teamcenter Integration Framework connector is established when the connector EJB is deployed into the web application server.

connector-configuration-name

Specifies the name of a Teamcenter Integration Framework connector configuration file.

The connector configuration file specifies the data source that the connector communicates with to obtain the data. The administrator creates and names this file when configuring connectors.

The configuration file name is optional. If you do not specify it, Teamcenter Integration Framework assumes it is the same as the bean name.

EXAMPLE

```
<conn:data-source-spec bean-name="EntBox"
   config-name="TcEntConfig"/>
```

NOTES

- The **data-source-spec** element is required for a data segment.
- A data segment can have only one **data-source-spec** element.
- See also *data-segment*.

default-value

PURPOSE

Supplies a default value for the attribute when the user inserts (creates) a business object instance in the data source or sources.

SYNTAX

<default-value>*default-value***</default-value>**

ATTRIBUTES

None.

EXAMPLE

```
<default-value>10</default-value>
```

NOTES

- The **default-value** element is optional.

  If you do not specify a value, Teamcenter Integration Framework does not supply a default value for the attribute.

  An attribute can have only one **default-value** element.

- If a form contains a blank for the attribute value when the business object is submitted, Teamcenter Integration Framework supplies the value defined in this element before inserting (creating) the business object in the data source.

- The **default-value** element applies only to an insert operation, not an update operation.

- See also *attribute* and *insert-able*.

default-color

PURPOSE

Specifies a color for displaying the data segment to a Teamcenter Integration Framework user, typically in the following hexadecimal format:

#*RRGGBB*

Replace *RR* with two digits specifying the amount of red; replace *GG* with two digits specifying the amount of green; replace *BB* with two digits specifying the amount of blue.

SYNTAX

<default-color>*color-ref***</default-color>**

ATTRIBUTES

None.

EXAMPLE

```
<default-color>#999966</default-color>
```

NOTES

- The **default-color** element is optional.

  A data segment can have only one **default-color** element.

- The **default-color** element is for demonstration purposes: to color-code aggregate business object information by data source location. The XSL defined for the user interface is the primary means to determine color in the user interface and can override this element.

- See also *data-segment*.

delete-able

PURPOSE

Determines whether Teamcenter Integration Framework users can remove business object instances from the data source defined for a data segment. If set to **true**, allows Teamcenter Integration Framework users to delete business object instances.

SYNTAX

<delete-able>*true-or-false***</delete-able>**

EXAMPLE

```
<delete-able>true</delete-able>
```

NOTES

- The **delete-able** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  A data segment can have only one **delete-able** element.

- To allow Teamcenter Integration Framework users to delete instances of an aggregate or multisource business object in each data source, you must specify **true** for the primary data segment and for each secondary data segment.

  However, you can permit users to delete instances in the primary data source, but not in all secondary data sources. The business object server deletes the instance in each data source, beginning with the primary data segment and continuing either until all segments are deleted or it encounters a data segment that does not have a **delete-able** element value defined as **true**.

- If you define the **delete-able** element value as **true** for a data segment, Teamcenter Integration Framework assumes that the data source defined for the data segment supports the deletion of instances.

- See also *data-segment*.

dynamic-attribute

PURPOSE

Defines a basic unit of information associated with a business object. The attribute value varies based on the values of other attributes as determined by a script.

SYNTAX

<dynamic-attribute name="*attribute-name***" >**
<label ... />
<type> ... </type>
<sort-key> ... </sort-key>
<script-file> ... </script-file>
<script-text> ... </script-text>
</dynamic-attribute>

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the elements in the order shown in the **dynamic-attribute** element syntax.

ATTRIBUTES

name

Specifies the business object attribute name.

Business object server requests reference the attribute by this name, which can be different from the name that the attribute is known by within the data source.

This name is required. The name can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces.

You can change this name for display to Teamcenter Integration Framework users with the **label** element.

EXAMPLES

```
<dynamic-attribute name="PartNumber">
      :
</dynamic-attribute>
<dynamic-attribute name="Revision">
      :
</dynamic-attribute>
```

NOTES

- An **attribute** or **dynamic-attribute** element is required for a data segment.

  A data segment can contain any number of **dynamic-attribute** elements.

  An attribute cannot have both **attribute** elements and **dynamic-attribute** elements.

- If the data source requires that a user enter a value for an attribute when creating a business object instance, you must specify the attribute within the business object definition.

- See also *label*, *type*, *sort-key*, *script-file*, and *script-text*.

dynamic-valid-value

PURPOSE

Displays valid values for the attribute obtained from attributes of business object instances.

SYNTAX

<dynamic-valid-value object="*business-object-name***"**
attribute="*attribute-name***"**
where-clause="*where-clause***" />**

ATTRIBUTES

object

Specifies the business object name whose instance contains valid values.

You must provide a business object name when defining dynamic values.

attribute

Specifies the attribute name for the business object defined in the **object** attribute. Teamcenter Integration Framework displays the attribute value as a valid value for the Teamcenter Integration Framework attribute you are defining.

You must provide an attribute name when defining dynamic values.

where-clause

Specifies the **where-clause** statement that is sent to the data source containing the business object to determine the attribute value for the business object instance.

The **where-clause** attribute is optional. If you do not specify a **where-clause** attribute, Teamcenter Integration Framework displays all possible entries from the specified attribute.

EXAMPLES

The following example identifies **employee** as the business object containing valid values and identifies the **EMPNO** attribute of **employee** as containing the valid values for the Teamcenter Integration Framework attribute you are defining:

```
<dynamic-valid-value object="employee" attribute="EMPNO"
   where-clause="EMPNO LIKE '0%'"/>
```

NOTES

- The **dynamic-valid-value** element is optional.

An attribute can have only one **dynamic-valid-value** elements.

An attribute cannot have both **valid-value** elements and a **dynamic-valid-value** element.

- Valid values are displayed as choices in the attribute box on the Teamcenter Integration Framework insert and update HTML forms. These forms do not allow users to enter values other than the displayed values.

  > Note:
  >
  > The Teamcenter Integration Framework business object server does not enforce valid values. Therefore, if your user interface allows users to enter invalid values, or if the values are sent directly to the business object server, they are accepted.

- See also *attribute*, *insert-able*, and *update-able*.

expand

PURPOSE

Specifies the relation to expand in a secondary data segment using attribute values supplied by other data segments.

SYNTAX

<expand relation="*relation-name***">**
  <util:where-clause> ... </util:where-clause>
  <attribute ...> or **<dynamic-attribute ... >**
  : **:**
  </attribute> **</dynamic-attribute>**
  :
</expand>

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the elements in the order shown in the **expand** element syntax.

ATTRIBUTES

relation

Specifies the business object relation to expand.

EXAMPLE

The following business object definition expands the **HasParts** relation in data segment **B**. This generates the bill of materials for the part in data segment **A**. The **Quantity** attribute is extracted for the relation objects from the data source.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--
bcprt
This software and related documentation are proprietary to Siemens Digital
Industries Software. COPYRIGHT 2005 UGS CORP.  ALL RIGHTS RESERVED
ecprt
-->
<business-object-definition name="TcEntBOM"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xmlns:conn="http://teamcenter.com/integrator/connection/2005-06"
                 xmlns:util="http://teamcenter.com/integrator/util/2005-06"
                 xsi:noNamespaceSchemaLocation="business-object-definition.xsd"
                 xmlns="http://teamcenter.com/integrator/bod/2005-06">
  <label default="TcEnt - BOM"></label>
  <data-source name="TcEntBOM">
    <data-segment name="A">
      <conn:data-source-spec bean-name="EntBox" config-name="TcEntConfig"/>
```

```
      <class-name>Assembly</class-name>
      <attribute name="OBID">
        <primary-key>true</primary-key>
        <hidden>true</hidden>
      </attribute>
      <attribute name="AssmPartNumber">
        <name-on-source>PartNumber</name-on-source>
        <search-able>true</search-able>
      </attribute>
      <attribute name="AssmRevision">
        <name-on-source>Revision</name-on-source>
        <search-able>true</search-able>
        <hidden>true</hidden>
      </attribute>
      <attribute name="AssmNomenclature">
        <name-on-source>Nomenclature</name-on-source>
        <search-able>true</search-able>
        <update-able>true</update-able>
      </attribute>
      <attribute name="AssmSequence">
        <name-on-source>Sequence</name-on-source>
        <search-able>true</search-able>
      </attribute>
      <attribute name="Immutable">
        <name-on-source>Immutable</name-on-source>
        <value-map-name>TcEnterpriseBooleans</value-map-name>
        <update-able>false</update-able>
      </attribute>
    </data-segment>
    <data-segment name="B">
      <multiple-results>true</multiple-results>
      <conn:data-source-spec bean-name="EntBox" config-name="TcEntConfig"/>
      <class-name>Assembly</class-name>
      <relation name="HasParts">
        <result-name>TcEntPart</result-name>
        <formal-name>PartsInAssembly</formal-name>
      </relation>
      <expand relation="HasParts">
        <util:where-clause>PartNumber = 'A.AssmPartNumber' and Revision =
'A.AssmRevision'
 and Sequence = 'A.AssmSequence'</util:where-clause>
        <attribute name="Qty">
          <name-on-source>Quantity</name-on-source>
        </attribute>
      </expand>
      <attribute name="CmptPartNumber">
        <name-on-source>PartNumber</name-on-source>
      </attribute>
      <attribute name="CmptRevision">
        <name-on-source>Revision</name-on-source>
      </attribute>
      <attribute name="CmptSequence">
        <name-on-source>Sequence</name-on-source>
      </attribute>
    </data-segment>
    <data-segment name="C">
      <multiple-results>true</multiple-results>
      <conn:data-source-spec bean-name="EntBox" config-name="TcEntConfig"/>
      <class-name>Part</class-name>
      <relation name="HasDescribingDocs">
```

```
        <result-name>TcEntGenericDocument</result-name>
        <formal-name>DocumentsDescribingPart</formal-name>
      </relation>
      <expand relation="HasDescribingDocs">
        <util:where-clause>PartNumber = 'B.CmptPartNumber' and Revision =
'B.CmptRevision'
 and Sequence = 'B.CmptSequence'</util:where-clause>
      </expand>
      <attribute name="DocName">
        <name-on-source>DocumentName</name-on-source>
      </attribute>
      <attribute name="DocType">
        <name-on-source>DocumentType</name-on-source>
      </attribute>
    </data-segment>
  </data-source>
</business-object-definition>
```

NOTES

- The **expand** element is optional within a data segment.

  You can specify only one **expand** element within a data segment.

- The **expand** element is valid only in secondary data segments in an aggregate business object definition.

- The **expand** element is valid only in business object definitions used with Teamcenter Enterprise connectors that implement the expand functionality.

- The **where-clause** (aggregate) element is required and specifies the data instance to expand. It refers to attributes defined in previous segments. It must uniquely specify a single object to expand and contain all key attributes defined for the data source class.

  You can specify only one **where-clause** (aggregate) element within the **expand** element.

- The **attribute** element is optional within an **expand** element.

  When you define an attribute within **expand**, Teamcenter Integration Framework uses the attribute from the relation object. When you define an attribute outside **expand**, Teamcenter Integration Framework uses the object attribute returned by expand.

  You can specify any number of **attribute** elements within **expand**.

- The **class-name** element in the data segment containing the **expand** element specifies the data instance class to expand.

- Only formal relations can be used in **expand** elements.

- You cannot use the **where-clause** (relationship) element and the **expand** element in the same data segment. However, you can use them in the same business object definition. For example, a data segment with a **where-clause** (relationship) element can be followed by a data segment with an **expand** element, followed by a data segment with another **where-clause** (relationship) element.

- To allow the expansion to return multiple results, you must add the **multiple-results** element to the data segment with the element value set to **true**.

formal-name

PURPOSE

Defines the name of a formal relationship, that is, a relationship modeled and maintained by a data source.

SYNTAX

<formal-name>*formal-name***</formal-name>**

ATTRIBUTES

None.

EXAMPLE

```
<formal-name>Contains</formal-name>
<formal-name>ContainedBy</formal-name>
```

NOTES

- The **formal-name** element is required for a formal relationship.

  Each formal relationship can have only one **formal-name** element.

- Do not define both a **formal-name** element and a **where-clause** element for the same relationship. When a **formal-name** element is defined, Teamcenter Integration Framework assumes the relationship is formal.

- See also *relation* and *result-name*.

hidden

PURPOSE

Determines whether the default Teamcenter Integration Framework user interface includes the attribute when displaying an object to the user in a table. If set to **true**, Teamcenter Integration Framework does not display the attribute containing the hidden element in the user interface.

SYNTAX

<hidden>*true-or-false***</hidden>**

ATTRIBUTES

None.

EXAMPLE

```
<hidden>true</hidden>
```

NOTES

- The **hidden** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  An attribute can have only one **hidden** element.

- Use the **hidden** element for multi-segment searches to include attributes that identify data returned in subsequent segments. This information is not useful to the user and does not need to be displayed.

- When the business object has an **insert-able** value of **true**, the default Teamcenter Integration Framework user interface displays hidden attributes in the insert form.

- Hidden attributes can have a **search-able** value of **true**.

- Hidden attributes cannot have an **update-able** value of **true**.

- See also *attribute*, *insert-able*, *search-able*, and *update-able*.

icon-name

PURPOSE

Associates a default icon image with the business object by specifying the universal resource locator (URL) that defines the icon location.

SYNTAX

<icon-name>*URL-icon-reference***</icon-name>**

ATTRIBUTES

name

EXAMPLE

```
<icon–name="http://localhost:80/icons/part.gif</icon-name>
<icon–name="http://host/icons/assembly.gif"</icon-name>
```

NOTES

- The **icon-name** element is optional.

  A data definition can have only one **icon-name** element.

- An icon is a small **jpg** or **gif** file that symbolically represents a business object.

- Teamcenter Integration Framework uses the icon element in displaying the business object information to the user.

- See also *business-object-definition*.

insert-able

PURPOSE

Determines whether Teamcenter Integration Framework users can create business object instances in the data source defined for the data segment. If set to **true**, allows Teamcenter Integration Framework users to create business object instances.

SYNTAX

<insert-able>*true-or-false***</insert-able>**

ATTRIBUTES

None

EXAMPLE

```
<insert-able>true</insert-able>
```

NOTES

- The **insert-able** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  A data segment can have only one **insert-able** element.

- To allow users to create instances of an aggregate or multisource business object in each data source, you must specify **true** for the primary data segment and for each secondary data segment.

  However, you can permit users to create instances in the primary data source, but not in all secondary data sources. The business object server creates an instance in each data source, beginning with the primary data segment and continuing either until all segments are created or it encounters a data segment that does not have an **insert-able** value defined as **true**.

- If you define the **insert-able** element value as **true** for a data segment, Teamcenter Integration Framework assumes that the data source defined for the data segment supports the creation of new instances.

- See also *data-segment*.

label

PURPOSE

Specifies the text displayed to the user for the name of a business object or the name of a business object attribute.

SYNTAX

<label ID="*text-identifier***" default="***text-string***" />**

ATTRIBUTES

ID

Specifies the identifier that maps a business object or attribute name to the localized text in the ID map XML file.

This attribute is optional. Specify this attribute only when you are using ID maps to display multiple text variations for various locales.

> Note:
>
> Do not specify the **ID** attribute if you are not using ID mapping. If you specify the **ID** attribute when no ID map exists, Teamcenter Integration Framework cannot find an ID and returns the business object definition file name or attribute by default.

default

Specifies text to display to the user for the business object name or business object attribute.

This attribute is optional. When you need to tailor a name or provide one localized name, specify the text to display in this attribute.

If you are using ID maps to map multiple text variations for various locales, specify this attribute to define a default label when either the identifier specified for the **ID** attribute does not exist in the ID map file or no display text exists for the identifier.

If you do not specify this attribute, Teamcenter Integration Framework uses the business object definition name (for a business object) or the name defined on the **attribute** element (for a business object attribute).

EXAMPLES

- The following examples use the **default** attribute of **label** to escape the naming convention for Teamcenter Integration Framework business objects and attributes:

```
<business-object-definition name="PartInventory">
      <label default="Part Inventory"/>
            :
</business-object-definition>

<attribute name="ThreeDimenImage">
      <label default="3D Image"/>
            :
</attribute>
```

- The following example defines a label for a business object. At runtime, Teamcenter Integration Framework finds the **Employee.name** ID in the **IDMap.xml** file in the datastore **i18n** folder defined by the user's locale and displays the corresponding localized name. If the ID does not exist in the ID map file, Teamcenter Integration Framework displays **Employee**.

  ```
  <business-object-definition name="Employee">
     <label ID="Employee.name"/>
  ```

- The following example defines a label for an attribute. At runtime, Teamcenter Integration Framework finds the **Employee.attr.EMPNO** ID in the **IDMap.xml** file in the datastore **i18n** folder defined by the user's locale. If the ID does not exist in the map, Teamcenter Integration Framework displays **Employee No.**:

  ```
  <attribute name="EMPNO">
     <label ID="Employee.attr.EMPNO" default="Employee No."/>
     <name-on-source>EMPNO</name-on-source>
     <primary-key>true</primary-key>
     <search-able>true</search-able>
  </attribute>
  ```

NOTES

- The **label** element is optional. If you do not specify the **label** element, Teamcenter Integration Framework displays the name defined for the **business–object–definition** or **attribute** elements.

  A business object can have only one **label** element; an attribute definition can have only one **label** element.

- Use the **label** element to display the name of any business object or business object attribute when that name cannot be accommodated within the Teamcenter Integration Framework naming convention (1 to 24 ASCII alphanumeric characters and underscores beginning with an alphabetical character and excluding embedded blank spaces).

- See also *attribute* and *business-object-definition*.

link

PURPOSE

Relates a business object either to other business objects or to a URL reference.

SYNTAX

```
<link name="link-name"
      href="URL-reference?|&"
      condition="condition-clause"
      encode="true-or-false"
      charset="encoding-character-set" >
  <label ... />
</link>
```

ATTRIBUTES

name

Specifies a name that uniquely identifies the link within Teamcenter Integration Framework. In the Teamcenter Integration Framework user interface, a named link displays a shortcut menu.

The **name** attribute is optional.

href

Specifies a URL reference that defines the link relationship.

Typically, the URL references an action component that performs a user interface function, such as displaying data or expanding a formal relationship. You can also use the URL to link to help text, another web site, an **ftp** site, and so forth. The URL must end with either a question mark character (**?**) or the ampersand text entity (**&**).

The **href** attribute is required.

> Note:
>
> Enter the URL reference in native characters. The browser encodes the HTTP reference; Teamcenter Integration Framework encodes in UTF-8 the parameters and values associated with getting action components or other user interface components. The URL you specify is appended to contain this encoding information. Therefore the URL must end with either a question mark (**?**) or **&** entry.

condition

Specifies a Boolean expression used to determine whether the link applies to an instance of a business object.

The **condition** attribute is optional.

The condition clause consists of condition terms (composed of an attribute reference, a comparison operator, and a value) in the following format (parentheses are not allowed):

attribute-reference operator value **[AND|OR** *attribute-reference operator value* **] ...**

attribute-reference

Specifies an attribute value of any previously defined data segment.

operator

Comparison operator used to compare the referenced attribute value with a fixed value. You can enter either a 2-character abbreviation or a symbol specified using XML entity substitution:

Operator	Abbreviation	Symbol
Equal	**eq**	**=**
Not equal	**ne**	**<>**
Less than	**lt**	**<**
Greater than	**gt**	**>**

Comparisons are made based on the type of attribute specified on the operator's left side. If the attribute is **string**, a string compare is performed. If the attribute is **int** or **double**, a numeric comparison is performed.

value

Specifies either a fixed value or an attribute value used as the reference for the comparison.

For the format of an attribute value, see the preceding description of *attribute-reference*.

You must enclose a string constant in single quotation marks.

encode

Specifies whether the parameters on a URL reference are encoded.

This attribute is optional. If you do not specify this attribute, Teamcenter Integration Framework encodes the parameters.

charset

Specifies a character set for encoding parameters on a URL reference.

This attribute is optional. If you do not specify this attribute, Teamcenter Integration Framework encodes the URL reference in UTF-8.

EXAMPLES

- The following example shows an unnamed, unconditional link using a **search** link call without the use of a **relation** definition:

```
<link
      href="http://localhost:7001//controller/search?Object=Inventory&Fields=1&a
mp;FieldName1=RespPerson&FieldValue1=this.EMPNO&">
</link>
```

- The following example shows a named, unconditional link using an **ID** element to map the link name to translated text in **IDMap.xml** files:

```
<link name="All Employees" href="search?Object=Employee&Fields=0">
   <label ID="SAMPLE.link.AllEmp"/>
</link>
```

- The following example shows a named, conditional link using an **expand** action component call to expand a **relation** definition:[1]

```
<link name="Contains" condition="Class eq 'Assembly'"
      href="expand?Object=Part&Relation=PartsIn
Assembly&Fields=1&FieldName1=PartNumber&FieldValue1=this.PartNumber&"/>
```

- The following example shows a named, unconditional link using an **expand** action component call to expand a **relation** definition:[2]

```
<link name="Described By"
      href="expand?Object=Part&Relation=DocumentsDescribingPart&Fields=
      1&FieldName1=PartNumber&FieldValue1=this.PartNumber

&?>&"/>
```

- The following example shows a simple, named, unconditional link to a help text HTML page:

```
<link name="Help" href="ObjectHelp/Part.html?"/>
```

> Note:
>
> **ObjectHelp** is an example only; it is not provided with Teamcenter Integration Framework.

NOTES

- The **link** element is optional.

  An attribute can have any number of **link** elements.

1 The link definition is the same for formal and informal relations.
2 The link definition is the same for formal and informal relations.

> Note:
>
> The Teamcenter Integration Framework user interface supports only 15 named links per attribute.

> Note:
>
> Although you can associate multiple links with any attribute in the business object, it can be more efficient to associate all named links for a business object with one attribute. The efficiency gained is based on how the user interface supports named links.
>
> For example, the Teamcenter Integration Framework user interface uses a JavaScript shortcut menu to display the named links. Each attribute value in the table has its own menu.
>
> Unnamed links are simple HTML references and do not require special processing. You can add any number of unnamed links to an attribute without incurring performance overhead.

- Teamcenter Integration Framework client applications use links to dynamically create data browsing capabilities linking one business object to another.
- A link can represent a either formal relationship modeled by a data source, such as a product knowledge management (PKM) system, or an informal relationship linking a business object to related information.
- See also *attribute* and *relation*.

multiple-results

PURPOSE

Indicates whether a secondary data segment that uses a **where-clause** (relationship) or **expand** element is allowed to return multiple data instances. If set to **true**, the **multiple-results** element specifies that multiple data instances are allowed to be returned from the query.

SYNTAX

<multiple-results>*true-or-false***</multiple-results>**

ATTRIBUTES

None.

EXAMPLE

```
<multiple_results>true</multiple_results>
```

NOTES

- The **multiple_results** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  A data segment can have only one **multiple_results** element.

- The **multiple_results** element is valid only in a secondary data segment.

- See also *data-segment*.

name-on-source

PURPOSE

Defines the attribute name as it is known within the data source.

You can specify a data name either within double quotation marks (quoted) or without (unquoted). Unquoted data names must begin with a letter and can include the following characters: letters **a** through **z** and **A** through **Z**, digits **0** through **9**, underscore (_), dollar sign (**$**), and number sign (**#**).

Quoted data names must begin with a letter and can include the following characters: letters **a** through **z** and **A** through **Z**, digits **0** through **9**, space ( ), ampersand (**&**), underscore (_), dash (**–**), greater than (**>**), less than (**<**), dollar sign (**$**), and number sign (**#**).

> Note:
>
> For the following special characters, you must use predefined XML text entities in quoted data names: double quotation mark (**"**), ampersand (**&**), greater than (**>**), and less than (**<**).

SYNTAX

<name-on-source>*data-name***</name-on-source>**

ATTRIBUTES

None.

EXAMPLE

- The following examples defines unquoted data names:

```
<name-on-source>PartNum</name-on-source>
<name-on-source>employee.frst</name-on-source>
```

- The following example defines a quoted data name:

```
<name-on-source>
    "&quot;EMP - & -&gt; &gt; &lt; NO&quot;"
</name-on-source>
```

  The quoted data name resolves to **'EMP – & –> > < NO'**.

NOTES

- The **name-on-source** element is optional.

If you do not specify a name for the attribute on the data source, Teamcenter Integration Framework uses the business object attribute name.

Each attribute can have only one **name-on-source** element.

- Some JDBC databases require that you define the table and column name containing the attribute. You can do this by specifying the table name and column name in the **name-on-source** element. For example:

```
<attribute name="FirstName">
   <name-on-source>"employee.frst"</name-on-source>
<attribute/>
```

This example specifies that the **FirstName** attribute is in the **employee** table and the **frst** column.

- Some JDBC drivers require that an SQL statement enclose in quotation marks the name of any attribute, table, or column containing a special character, such as a dash (-). For example, for an SQL statement to parse properly with these JDBC drivers, the **Part-No** column name must be enclosed within quotation marks as follows:

```
"Part-No"
```

To specify quotation marks within the **name-on-source** element, you must use the XML **"** text entity. For example:

```
<name-on-source>"&quot;Part-No&quot;"</name-on-source>
```

> Note:
>
> Although some JDBC drivers require quotation marks enclosing names with special characters, other JDBC drivers do not allow quotation marks. If you are not sure which JDBC driver is required, test your business object definitions.

- See also *attribute*.

param

PURPOSE

Allows the extension of business object definitions by specifying parameters.

SYNTAX

<util:param name="*name***" value="***value***" />**

ATTRIBUTES

name

Specifies the name of a parameter. This attribute is required. The following parameter names are part of standard Teamcenter Integration Framework:

max_objects

Specifies the maximum number of objects returned from a query. Define this parameter name only within a **business-object-definition** element.

merge-duplicate-cells

Causes Teamcenter Integration Framework to merge the displayable values for cells in successive rows that contain the same value in each cell. Hidden values do not affect whether the cells are merged or not. You may also define this parameter name within a **data-segement** element. The following shows possible results for a message result query with this parameter not defined or set to **false**:

MessageID	Type	Reactor name	Result	Activity
TcEngV10infodba_6716556	BOM transfer	BOM transfer	Failed	BOM transfer
TcEngV10infodba_6716556	BOM transfer	BOM transfer	Failed	BOM download
TcEngV10infodba_6716556	BOM transfer	BOM transfer	Failed	BOM upload
TcEngV10infodba_6716569	BOM transfer	BOM transfer	Started	BOM transfer

The results with the **merge-duplicate-cells** value set to **true** for the **MessageID** attribute:

MessageID	Type	Reactor name	Result	Activity
TcEngV10infodba_6716556	BOM transfer	BOM transfer	Failed	BOM transfer
	BOM transfer	BOM transfer	Failed	BOM download
	BOM transfer	BOM transfer	Failed	BOM upload
TcEngV10infodba_6716569	BOM transfer	BOM transfer	Started	BOM transfer

The results with the **merge-duplicate-cells** value set to **true** for the **data-segement** element:

MessageID	Type	Reactor name	Result	Activity
TcEngV10infodba_6716556	BOM transfer	BOM transfer	Failed	BOM transfer
				BOM download
				BOM upload
TcEngV10infodba_6716569	BOM transfer	BOM transfer	Started	BOM transfer

Your enterprise can also define custom parameter names in a Teamcenter Integration Framework Java customization.

value

Specifies the parameter value. This attribute is required.

For use with **max_objects**, you can specify any positive integer or zero (0). A positive integer limits the objects a query returns to the number specified. Zero returns all objects found. For use with **merge-duplicate-cells**, you can specify either **true** or **false**.

The valid values for a custom parameter are defined in a Teamcenter Integration Framework Java customization.

EXAMPLES

- The following example specifies that a maximum of 1188 objects are returned in a query for message results with duplicate cells in successive rows containing the same message ID merged:

```
<business-object-definition name="MsgResult">
   <util:param name="max_objects" value="1188"/>
    <sort–direction>ascending</sort–direction>
   <data–source name="MsgResult">
               <data-segment name=”Result”>
                   :
                   <attribute name=MessageID”>
                       <util:param name="merge-duplicate-cells"
value="true"/>
                  :
                   </attribute>
               </data-segment>
    :
   </data–source>
   :
</business-object-definition>
```

- The following example specifies a custom parameter named **Main** and specifies the value of **Main** as **true**:

```
<util:param name="Main value="true"/>
```

- The following example specifies a custom parameter named **xref** and specifies the value of **xref** as **pkmComponent**:

```
<util:param name="xref" value="pkmComponent"/>
```

NOTES

- The **param** element is optional. If you do not specify the **param** element with the **max_objects** parameter defined, a maximum of 50 objects are returned in a query.

  The **business-object-definition**, **data-source**, **data-segment**, **relation**, and **attribute** elements can each have any number of **param** elements. Define the **max_objects** parameter only when the **param** element is specified for a **business-object-definition** element.

- If you do not specify the **param** element with the **merge-duplicate-cells** parameter, Teamcenter Integration Framework does not merge cells in rows with duplicate attribute values.

- Using the **param** element with any **name** value other than **max_objects** or **merge-duplicate-cells** requires a customization of Teamcenter Integration Framework using published Java APIs that return

the name and values for a specific business object and XML element. If the customization is not implemented, the **param** element is ignored.

For more information, see the **getCustomParams** and **getCustomParamValue** method descriptions in the JavaDoc for the **Client** class.

- See also *attribute*, *business-object-definition*, *data-segment*, *data-source*, and *relation*.

primary-key

PURPOSE

Indicates whether the attribute is required to determine instance uniqueness. The combination of all primary key values guarantees a unique business object instance. If set to **true**, uses the attribute value to provide business object instance uniqueness.

SYNTAX

<primary-key>*true-or-false***</primary-key>**

ATTRIBUTES

None.

EXAMPLE

```
<primary-key>true</primary-key>
```

NOTES

- The **primary-key** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  An attribute can have only one **primary-key** element.

- The **primary-key** element applies only to attributes in primary data segments. If you specify **primary-key** in a secondary data segment, Teamcenter Integration Framework ignores it.

- An attribute cannot have both a **primary-key** value of **true** and an **update-able** value of **true**.

- See also *attribute* and *update-able*.

relation

PURPOSE

Defines a formal or informal relationship between two business objects:

- A *formal relationship* is maintained by a data source, for example, a product data management (PDM) system.
- An *informal relationship* is a foreign key join.

SYNTAX

Formal Relationship

```
<relation name="relation-name">
    <util:param ... />
      :
    <util:param ... />
    <result-name> ... </result-name>
    <formal-name> ... </formal-name>
</relation>
```

Informal Relationship

```
<relation name="relation-name">
    <util:param ... />
      :
    <util:param ... />
    <result-name> ... </result-name>
    <util:where-clause> ... </util:where-clause>
</relation>
```

> Note:
>
> The **business-object-definition.xsd** schema requires that you specify the elements in the order shown in the **relation** element syntax.

ATTRIBUTES

name

Specifies the business object relationship name. Business object server requests reference the relationship by this name, which can be different from the name by which the relationship is known within the data source.

The name can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces.

EXAMPLE

The following examples show a formal and informal relationship:

```
<relation name="HasParts">
   <result-name>Part</result-name>
   <formal-name>PartsInAssembly</formal-name>
</relation>

<relation name="resultById">
   <result-name>MsgResult</result-name>
   <util:where-clause>MessageID = 'this.MessageID'
   </util:where-clause>
 </relation>
```

NOTES

- The **relation** element is optional.

  A data segment can contain any number of **relation** elements.

- See also *param*, *result-name*, *formal-name*, *where-clause (relationship)*, and *data-segment*.

required-for-insert

PURPOSE

Specifies whether the user must supply a value for this attribute when inserting (creating) a business object instance in the data source.

SYNTAX

<required-for-insert>*true-or-false***</required-for-insert>**

ATTRIBUTES

value

Specifies whether the attribute requires a value when the user is inserting a business object instance in the data source.

EXAMPLE

```
<required-for-insert>true</required-for-insert>
```

NOTES

- The **required-for-insert** element is optional.

  An attribute can have only one **required-for-insert** element.

- See also *attribute*, *default-value* and *valid-value*.

result-name

PURPOSE

Defines the business object definition (BOD) name that describes the type of business object instances returned when the relationship is expanded. The element value can be from 1 to 24 ASCII alphanumeric characters and underscores. The first character must be alphabetical. Do not include embedded blank spaces.

SYNTAX

<result-name>*business-object-name***</result-name>**

ATTRIBUTES

None.

EXAMPLE

```
<result–name>Part</result-name>
<result–name<Document</result-name>
```

NOTES

- The **result-name** element is required for a relationship definition.

  A relationship definition can have only one **result-name** element.

- See also *relation*.

script-file

PURPOSE

Defines the attribute value in terms of other attribute values by providing a script used to determine the value. The element identifies the script file's full path and file name. The attribute value is the result of operations performed in a Tcl script.

SYNTAX

<script-file>*path-to-and-script-filename***</script-file>**

ATTRIBUTES

None.

EXAMPLES

The following example defines the **CatComp** attribute as the result of a Tcl script contained in the **comp.tcl** file:

```
<dynamic-attribute name="CatComp">
   <script–file>c:\\globalservices\runtime\bos\comp.tcl
   </script–file>
</dynamic-attribute>
```

NOTES

- The **script-file** element is optional for a dynamic attribute definition.

  An attribute definition can have only one **script-file** element. A business object definition can have multiple **script-file** elements within attribute definitions.

- For more information on Tcl scripts, visit **https://www.activestate.com/**.

- Teamcenter Integration Framework evaluates Tcl scripts after receiving the objects matching a query and before returning the objects to the query originator. Teamcenter Integration Framework runs each Tcl script in the business object definition for each business object in the query results. The Tcl scripts have access to all business object attributes in the form of Tcl string variables. Each attribute is represented by a variable with the same name, defined as the attribute value.

- Teamcenter Integration Framework passes attributes to the TCL interpreter without segment prefixes. Therefore, you cannot prefix attribute references with segment names.

- The Tcl script syntax is not validated with the business object definition. It is validated when the script is evaluated.

- For multisource business object definitions, you must define the same attributes for each **data-source** element. However, if you include a **script-file** element for an attribute in one **data-source** element, you need not include a **script-file** element for the same attribute in the other **data-source** elements. The value for the attribute in one data source can be calculated by a script, while the value for the same attribute in a different data source can retrieved directly from the database.

  If you do provide a **script-file** element for the attribute in another **data-source** element, the Tcl script need not be the same. For example, you could provide a different Tcl script for one attribute if one data source is missing needed fields.

- Tcl scripts do not work with Teamcenter Integration Framework **compute** functions.

- See also *attribute*.

script-text

PURPOSE

Defines the attribute value in terms of other attribute values by providing a Tcl script within the element. The attribute value is the result of operations performed in a Tcl script.

SYNTAX

<script-text>
Tcl-script
</script-text>

ATTRIBUTES

None.

EXAMPLES

- The following example defines the **TotalComp** attribute in terms of three numeric attributes defined in the same business object definition, **Salary**, **Bonus**, and **Comm**. The Tcl script, provided in the business object definition file, returns the **TotalComp** attribute value as the sum of the numeric attribute values.

```
<attribute name="TotalComp">
   <script–text>
      expr $Salary + $Bonus + $Comm
   </script–text>
</attribute>
```

- The following example defines the **WholeName** attribute in terms of three attributes defined in the same business. The Tcl script returns the **WholeName** attribute value formed by concatenating the object definition's **FirstName**, **MI**, and **LastName** attribute values.

```
<attribute name="WholeName">
    <label default ="Name"/>
    <script–text>
       concat $FirstName $MI $LastName
    </script–text>
</attribute>
```

NOTES

- The **script-text** element is optional for an attribute definition.

An attribute definition can have only one **script-text** element. A business object definition can have multiple **script-text** elements within attribute definitions.

- For more information on Tcl scripts, visit **https://www.activestate.com/**.
- Teamcenter Integration Framework evaluates Tcl scripts after receiving the objects matching a query and before returning the objects to the query originator. Teamcenter Integration Framework runs each Tcl script in the business object definition for each business object in the query results. The Tcl scripts have access to all business object attributes in the form of Tcl string variables. Each attribute is represented by a variable with the same name, defined as the attribute value.
- Teamcenter Integration Framework passes attributes to the TCL interpreter without segment prefixes. Therefore, you cannot prefix attribute references with segment names.
- The Tcl script syntax is not validated with the business object definition. It is validated when the script is evaluated.
- For multisource business object definitions, you must define the same attributes for each **data-source** element. However, if you include a **script-text** element for an attribute in one **data_source** element, you need not include a **script-text** element for the same attribute in the other **data_source** elements. The value for the attribute in one data source can be calculated by a script, while the value for the same attribute in a different data source can retrieved directly from the database.

  If you do provide a **script-text** element for the attribute in another **data-source** element, the Tcl script need not be the same. For example, you could provide a different Tcl script for one attribute if one data source is missing needed fields.
- Tcl scripts do not work with Teamcenter Integration Framework **compute** functions.
- See also *attribute*.

search-able

PURPOSE

Indicates whether the attribute is included in search criteria. If set to **true**, includes the attribute in search criteria.

SYNTAX

<search-able>*true-or-false***</search-able>**

ATTRIBUTES

None.

EXAMPLE

```
<search-able>true</search-able>
```

NOTES

- The **search-able** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  An attribute can have only one **search-able** element.

- The **search-able** element applies only to attributes in the first data source's primary data segment. If you specify **search-able** in either a secondary data segment or the primary segment of a secondary data source, Teamcenter Integration Framework ignores it.

- See also *attribute*.

sort-direction

PURPOSE

Specifies the order that Teamcenter Integration Framework uses to sort attribute values returned from the data sources as the result of a query. Valid values are **ascending** or **descending**.

SYNTAX

<sort-direction>*ascending-or-descending***</sort-direction>**

ATTRIBUTES

None.

EXAMPLE

The following example defines ascending sorting order for the **MsgResult** business object's attributes:

```
<business-object-definition name="MsgResult">
  <sort-direction>ascending</sort-direction>
:
</business-object-definition>
```

Each attribute to be sorted must be defined using the **sort-key** element. For an example using both **sort-direction** and **sort-key**, see *sort-key*.

NOTES

- When you specify **sort-direction**, you must also specify the **sort-key** element to define the attributes sorted and their priorities.
- The **sort-direction** element is optional.

  If you do not specify **sort-direction**, but do specify **sort-key**, Teamcenter Integration Framework returns attribute values in ascending order.

  A business object data definition can have only one **sort-direction** element.
- Sorting attributes requires custom Java code that implements the **sort-direction** element. If your enterprise has not created the required customization, Teamcenter Integration Framework ignores the **sort-direction** element.
- Attributes with a type specified in the business object definition are sorted according to the **type**. For example, if the attribute type is **string**, the values are sorted alphabetically. If the attribute type is **int**, the values are sorted numerically.

- Attribute value sorting is also implemented using action components:
  - The search and search form action components installed with Teamcenter Integration Framework allow the user to specify the attribute sort order as part of a query.
  - Java customizers can create custom search and search form action components that allow the user a choice for sorting attributes.

  Note:

  Sorting implemented in an action component is independent of these sorting elements and should not be used in conjunction with the **sort-direction** and **sort-key** elements.

- See also *business-object-definition* and *sort-key*.

sort-key

PURPOSE

Specifies whether Teamcenter Integration Framework sorts the values for this attribute when returning query results from data sources; defines the attribute's sorting priority when you define sorting for multiple attributes. You specify an integer from **1** to any number. The integer represents the attribute's priority when Teamcenter Integration Framework sorts the results of a query. Specifying **1** indicates the highest priority.

SYNTAX

<sort-key>*integer***</sort-key>**

ATTRIBUTES

None.

EXAMPLES

The following example sorts the **Result**, **ReactorName**, and **ProcessedCount** attribute values in ascending order. Teamcenter Integration Framework sorts the **Result** attribute values first, next sorts the **ReactorName** attribute values, and sorts the **ProcessedCount** attribute values last. The **DeliveredDate** and **SequenceNo** attribute values are not sorted.

```
<business-object-definition name="MsgResult">
  <sort-direction>ascending</sort-direction>
    :
        <attribute name="ReactorName">
           <search-able>true</search-able>
     <sort-key>2</sort-key>
        </attribute>
        <attribute name="DeliveredDate"/>
        </attribute>
        <attribute name="Result" >
           <search-able>true</search-able>
     <sort-key>1</sort-key>
        </attribute>
        <attribute name="SequenceNo">
           <name-on-source>SequenceNumber</name-on-source>
        </attribute>
        <attribute name="ProcessedCount"/>
           <type>int</type>
     <sort-key>3</sort-key>
        </attribute>
    :
</business-object-definition>
```

NOTES

- The **sort-key** element requires custom Java code.
- The **sort-key** element is optional. If you do not specify **sort-key**, Teamcenter Integration Framework does not sort the values returned for the attribute (it returns attribute values in the order received from the data sources).

  An attribute can have only one **sort-key** element.
- The **sort-direction** element defines the sorting order.
- A data definition can have **sort-key** elements defined for multiple attributes.
- See also *business-object-definition* and *sort-direction*.

type

PURPOSE

Defines the type of data that the attribute represents.

SYNTAX

<type>*data-type***</type>**

The element must contain a supported data type. Supported data type values are:

Value	Description
bigint	Big numeric data
date	Date information
double	Double-size fractional data
float	Fractional data
int	Numeric data
smallint	Small numeric data
string	Character data
time	Time information
timestamp	Timestamp information

ATTRIBUTES

None.

EXAMPLE

```
<type>int</type>
<type>date</type>
```

NOTES

- The **type** element is optional.

  If you do not specify a data type, Teamcenter Integration Framework uses **string**.

  An attribute can have only one **type** element.

- See also *attribute*.

update-able

PURPOSE

Determines whether a Teamcenter Integration Framework user can update the attribute in the data source. If the element value is set to **true**, a Teamcenter Integration Framework client can update the attribute.

SYNTAX

<update-able>*true-or-false***</update-able>**

ATTRIBUTES

None.

EXAMPLE

```
<update-able>true</update-able>
```

NOTES

- The **update-able** element is optional.

  If you do not specify a value, Teamcenter Integration Framework uses **false**.

  An attribute can have only one **update-able** element.

- The **update-able** element applies to attributes in both primary and secondary data segments.

  If a secondary data segment has no data matching the **where-clause** element, Teamcenter Integration Framework stops processing the update for the remaining secondary segments.

- An attribute cannot have both a **primary-key** and an **update-able** value of **true**.

- An attribute cannot have both a **hidden** and an **update-able** value of **true**.

- When the **update-able** element value is **true**, the HTML forms generated for the business object allow the user to edit the attribute field.

- See also *attribute*, *data-segment*, *primary-key*, and *where-clause (aggregate)*.

valid-value

PURPOSE

Specifies a static string as a value for the attribute you are defining that Teamcenter Integration Framework displays as a valid value for the attribute. You can specify a blank as a valid value using either an empty string or a space:

```
value=""
or
value=" "
```

SYNTAX

<valid-value>*string***</valid-value>**

ATTRIBUTES

None.

EXAMPLES

The following example defines two static valid values for this attribute:

```
<valid-value>Yes</valid-value>
<valid-value>No</valid-value>
```

NOTES

- The **valid-value** element is optional.

  An attribute can have multiple **valid-value** elements.

  An attribute cannot have both **valid-value** elements and **dynamic-valid-value** elements.

- Valid values are displayed as choices in the attribute field of the Teamcenter Integration Framework sample insert and update HTML forms. These forms do not allow users to enter values other than the displayed values.

  > Note:
  >
  > The Teamcenter Integration Framework business object server does not enforce valid values. Therefore, if your user interface allows users to enter invalid values, or if the values are sent directly to the business object server, they are accepted.

- If you define a blank value as valid, and a user overwrites an existing numeric value with a blank on the Teamcenter Integration Framework sample update form, Teamcenter Integration Framework refuses to accept the value and displays an error message.

- See also *attribute*, *insert-able*, and *update-able*.

value-map-name

PURPOSE

Specifies the name of a value map used to map attribute values for a Teamcenter Integration Framework business object to data source values. This value is defined in the **ValueMaps.jaxb** file by the **value-map** element's **name** attribute (see *value-map* in *Attribute value map configuration elements*.

SYNTAX

<value-map-name>*map-file-name***</value-map-name>**

ATTRIBUTES

None.

EXAMPLES

- The following example defines **testmap** as the value map containing data source values for the attribute:

  ```
  <value-map-name>testmap</value-map-name>
  ```

NOTES

- The **value-map** element is optional.

  An **attribute** element can have only one **value-map** definition.

- See also *attribute*.

where-clause (aggregate)

PURPOSE

Specifies the secondary data source instance to be combined with the primary data source to create an aggregate business object. Teamcenter Integration Framework sends the clause string to a secondary data source to determine the instance to be used for data aggregation. It can reference the attribute values of any previously defined data segment.

SYNTAX

<util:where-clause>*string***</util:where-clause>**

string

Specifies the clause sent to a secondary data source to determine the instance to be used for data aggregation. It can reference the attribute values of any previously defined data segment.

The result returned from the where clause must evaluate to a single data instance. String references must be enclosed in single quotation marks. Operators, including the equal sign, must be surrounded by spaces. The following operators are supported:

```
<
<=
>
>=
=
!=
<>
*
/
+
-
```

ATTRIBUTES

None.

EXAMPLE

```
<data-segment name="A">
   :
   <attribute name="WorkDept">
      :
   </attribute>
   <attribute name="FName">
      :
   </attribute>
```

```
    <attribute name="LName">
       :
    </attribute>
  :
</data-segment>
<data-segment name="B">
  :
  <util:where-clause>DEPTNO = 'A.WorkDept'</util:where-clause>
     :
  <attribute name="MGRNO">
     :
  </attribute>
</data-segment>
<data-segment name="C">
  :
  <util:where-clause>FirstName = 'A.FName' AND LastName = 'A.LName'
     AND EMPNO = 'B.MGRNO'</util:where-clause>
  :
</data-segment>
```

NOTES

- The **where-clause** element is optional, but at least one **where-clause** or **expand** element is required for a secondary data segment.

  A data segment can have only one **where-clause** or **expand** element.

- A **where-clause** element must not be defined for a primary data segment.

- See also *data-segment* and *expand*.

where-clause (relationship)

PURPOSE

Determines the instances of the relationship result objects when an informal (foreign key) relationship is expanded.

SYNTAX

<util:where-clause>*attribute-name operator value***</util:where-clause>**

attribute-name

References an attribute in the result object's primary data segment. Only attributes defined within the result object's primary data segment can be referenced.

operator

Specifies an SQL comparison operator. Operators, including the equal sign, must be surrounded by spaces.

value

Specifies either a fixed value or a local attribute reference.

String references must be enclosed in single quotation marks.

A local attribute reference obtains the value of an attribute. You can reference only attributes within the data segment you are defining. A local attribute reference has the following format:

this.*attribute-name*

ATTRIBUTES

None.

EXAMPLE

```
<relation name="mgr">
   <result-name>Employee</result-name>
   <util:where-clause>EMPNO = 'this.MGRNO'</util:where-clause>
</relation>
```

NOTES

- The **where-clause** element is required for an informal (foreign key) relationship.

  An informal relationship can have only one **where-clause** element.

- You cannot define both a **where-clause** element and a **formal-name** element for the same relationship. When a **formal-name** element is defined, Teamcenter Integration Framework assumes the relationship is formal.

- The **where-clause** element clause is applied to the resulting business object; therefore, you can specify a mixture of references to the resulting business object attributes as well as the current business object attributes.

- See also *relation* and *result-name*.

Attribute value map configuration elements

Attribute value map file

Teamcenter Integration Framework includes an attribute value map file that is an example of mapping values from different data sources. You can use this file as an example for mapping data from multiple data sources onto single business object values. Each map is identified by a name, which is then used in the attribute definition of each business object.

```
<-- ValueMaps file -->
<?xml version="1.0" encoding="UTF-8"?>
<value-map-config>
   <value-map name="EmployeeTitlemapping">
      <ignore-case>true</ignore-case>
      <mapping BOS-value="Field Representative" source-value="FIELDREP"/>
      <mapping BOS-value="Sales Representative" source-value="SALESREP"/>
      <mapping BOS-value="Clerk" source-value="CLERK   "/>
      <mapping BOS-value="Designer" source-value="DESIGNER"/>
      <mapping BOS-value="Manager" source-value="MANAGER "/>
      <mapping BOS-value="Analyst" source-value="ANALYST "/>
      <mapping BOS-value="President" source-value="PRES    "/>
      <mapping BOS-value="Operator" source-value="OPERATOR"/>
   </value-map>
   <value-map name="MakeBuyIndicatorMTI">
      <mapping BOS-value="AccMake" source-value="PsmIndMake"/>
      <mapping BOS-value="AccBuy" source-value="PsmIndBuy"/>
   </value-map>
   <value-map name="MakeBuyIndicatorSAP">
      <mapping BOS-value="AccMake" source-value="E"/>
      <mapping BOS-value="AccBuy" source-value="F"/>
   </value-map>
   <value-map name="MakeBuyIndicatorOMFG">
      <mapping BOS-value="AccMake" source-value="1"/>
      <mapping BOS-value="AccBuy" source-value="2"/>
   </value-map>
</value-map-config>
```

ignore-case

PURPOSE

Instructs Teamcenter Integration Framework to ignore lettercase when mapping data source values to business object values. Use this element when mapping values for a case-insensitive database. If set to **true**, specifies case-insensitive value mapping.

SYNTAX

<ignore-case>*true-or-false***</ignore-case>**

ATTRIBUTES

None.

EXAMPLE

The following example instructs Teamcenter Integration Framework to ignore letter case when mapping data source values to business object values:

```
<ignore-case>true</ignore-case>
```

NOTES

- The **ignore-case** element is optional. If you do not specify this element, value mapping is case-sensitive.

  A **value-map** element can have only one **ignore-case** element.

- When you specify case-insensitive mapping, Teamcenter Integration Framework sends data source values in uppercase.

mapping

PURPOSE

Defines mapping of one value. Mapping applies in both directions of data instances exchanged between Teamcenter Integration Framework and the data source.

SYNTAX

<mapping BOS-value="*business-object-value***" source-value="***data-source-value***" />**

ATTRIBUTES

BOS-value

Specifies the value for the Teamcenter Integration Framework business object.

This attribute is required.

source-value

Specifies the value for the data source object.

When the data source is case-insensitive, it returns values that match both the uppercase and lowercase versions of this value. By default, Teamcenter Integration Framework is case-sensitive. If you want Teamcenter Integration Framework to recognize uppercase and lowercase versions of this value returned by the database, you must specify the **ignore-case** element in the value map.

This attribute is required.

EXAMPLE

The following example defines the Teamcenter Integration Framework business object attribute value as **accval1** and the data source attribute value as **val1**:

```
<mapping BOS-value="accval1" source-value="val1"/>
```

NOTES

One **mapping** element is required within a **value-map** element.

A **value-map** element can have multiple **mapping** elements.

value-map

PURPOSE

Defines one value map identified by name.

SYNTAX

<value-map name="*value-map-name***">**
<ignore-case> ... </ignore-case>
<mapping> ... </mapping>
<mapping> ... </mapping>
:
</value-mapp>

> Note:
>
> The **value-map.xsd** schema requires that you specify the elements in the order shown in the **value-map** element syntax.

ATTRIBUTES

name

Specifies the attribute value map name. This name is used in the business object definition to identify the value map for an attribute.

This attribute is required.

EXAMPLE

The following example defines a value map named **testmap**:

```
<value-map name="testmap">
      <mapping BOS-value="accval1" source-value="val1"</mapping>
      <mapping BOS-value="accval2" source-_value="val2"</mapping>
      <mapping BOS-value="accval3" source-value="val3"</mapping>
</value-map>
```

NOTES

- One **value-map** element is required in the **ValueMaps.jaxb** file.

  A **ValueMaps.jaxb** file can have multiple **value-map** elements.

- See also *value-map-name*.

value-map-config

PURPOSE

Defines a list of data source value maps for a business object attribute. This is the **ValueMaps.jaxb** file's root element.

SYNTAX

```
<value-map-config>
    <value-map> ... </value-map>
        :
    <value-map> ... </value-map>
    <value-map> ... </value-map>
        :
    <value-map> ... </value-map>
        :
</value-map-config>
```

ATTRIBUTES

None.

EXAMPLE

See *Attribute value map file*.

NOTES

One **value-map-config** element is required in each **ValueMaps.jaxb** file.

A **ValueMaps.jaxb** file can have only one **value-map-config** element.

ID map configuration elements

Sample ID maps

Teamcenter Integration Framework includes sample ID map files. The sample **IDMap.xml** files map localized names for the sample business object definitions. The following listings show sample map files for the **de_DE** locale and **fr_FR** locales.

```
<!-- Sample IDMap file for de_DE locale -->
<?xml version="1.0" encoding="UTF-8"?>
<IDMap>
   <TextID name="Employee.name"              text="Angestellte"/>
   <TextID name="Employee.attr.EMPNO"        text="Angestellte Nr."/>
```

```
   <TextID name="Employee.attr.FirstName"   text="Vorname"/>
   <TextID name="Employee.attr.LastName"    text="Zuname"/>
   <TextID name="Employee.attr.MI"          text="Anfänglich"/>
   <TextID name="Employee.attr.WorkDept"    text="Abteilung"/>
   <TextID name="Employee.attr.PhoneNo"     text="Telefonnummer"/>
   <TextID name="Employee.attr.Title"       text="Titel"/>
   <TextID name="Employee.attr.Sex"         text="Geschlecht"/>
   <TextID name="Employee.attr.BirthDate"   text="Geburtstag"/>
   <TextID name="Employee.attr.Salary"      text="Gehalt"/>
   <TextID name="Employee.attr.Bonus"       text="Prämie"/>
   <TextID name="Employee.attr.Comm"        text="Provision"/>
</IDMap>

<!-- Sample IDMap file for fr_FR locale -->
<?xml version="1.0" encoding="UTF-8"?>
<IDMap>
   <TextID name="Employee.name"             text="Employée"/>
   <TextID name="Employee.attr.EMPNO"       text="Employée No."/>
   <TextID name="Employee.attr.FirstName"   text="Prénom"/>
   <TextID name="Employee.attr.LastName"    text="Dernier nom"/>
   <TextID name="Employee.attr.MI"          text="Initiale"/>
   <TextID name="Employee.attr.WorkDept"    text="Département"/>
   <TextID name="Employee.attr.PhoneNo"     text="Télephone No."/>
   <TextID name="Employee.attr.Title"       text="Titre"/>
   <TextID name="Employee.attr.Sex"         text="Sexe"/>
   <TextID name="Employee.attr.BirthDate"   text="Anniversaire"/>
   <TextID name="Employee.attr.Salary"      text="Salarie"/>
   <TextID name="Employee.attr.Bonus"       text="Prime"/>
   <TextID name="Employee.attr.Comm"        text="Commission"/>
</IDMap>
```

IDMap

PURPOSE

Defines an ID map; the **IDMap.xml** file's root element.

SYNTAX

```
<IDMap>
    <TextID .... />
        :
     <TextID .... />
</IDMap>
```

ATTRIBUTES

None.

EXAMPLE

See *Sample ID maps*.

NOTES

One **IDMap** element is required in each **IDMap.xml** file.

An **IDMap.xml** file can have only one **IDMap** element.

TextID

PURPOSE

Maps the identifier defined in the business object definition **label** or **links** element to localized text displayed as the business object name, attribute name, or named link.

SYNTAX

<TextID name="*label-ID***" text="***translated-text***"/>**

ATTRIBUTES

name

Specifies the **ID** attribute value from the **label** or **links** element in the business object definition.

This attribute is required.

text

Specifies the localized text that Teamcenter Integration Framework displays as the business object name, attribute name, or link. You can enter any characters from any character sets supported by the Apache Xerces parser.

This attribute is required.

EXAMPLES

- The following example maps the **Employee.attr.FirstName** identifier (specified on a **label** element) to the text to display for this attribute in the **de_DE** locale:

```
<TextID name="Employee.attr.FirstName" text="Vorname"/>
```

- The following example maps the **Employee.attr.FirstName** identifier (specified on a **label** element) to the text to display for this attribute in the **fr_FR** locale:

```
<TextID name="Employee.attr.FirstName" text="Prénom"/>
```

NOTES

This element is required for an ID map.

An ID map can have any number of **TextID** elements.