SIEMENS

# Teamcenter Integration Framework

tcif1 • 3.2

# Contents

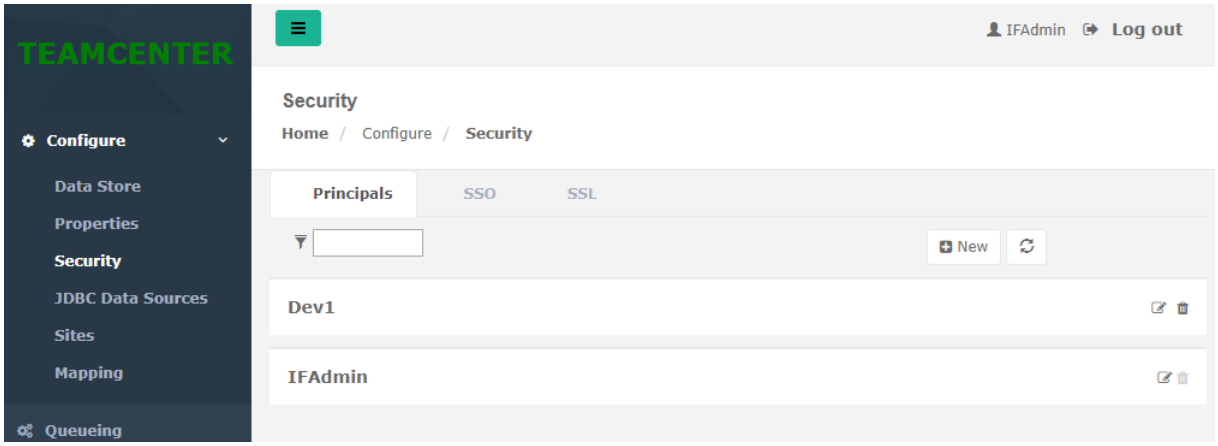
# Chapter 1: What's new in Teamcenter Integration Framework 3.2

# Chapter 1: What's new in Teamcenter Integration Framework 3.2

## Improved usability

Based on customer feedback, several parts of Teamcenter Integration Framework have been updated to improve ease of use and security, including:

- Creating and editing properties
- Managing Teamcenter Integration Framework users
- Managing data sources
- Managing sites exchanging data
- Creating and editing data transformation mappings
- Monitoring transaction activity




## Customize tracking of transaction status

You can now define custom message properties to enhance transaction status messages generated during Teamcenter Integration Framework processing. Customizing status messages can provide visibility into complicated transactions by making a broader set of transaction statuses available for monitoring than those available by default. You can programmatically retrieve and process this status information as described in Track activity status. Activity Status columns can also be customized for monitoring these properties.

# Chapter 2: Teamcenter Integration Framework overview

# Chapter 2: 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 Teamcenter Compatibility Matrix.

### 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.

# Chapter 3: Installing and upgrading Teamcenter Integration Framework

# Chapter 3: 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.

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 | | |
| 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 help collection available on the Siemens PLM Software Doc Center.

Use the following steps to install Teamcenter Integration Framework:

1. Download both the Teamcenter Foundation and Teamcenter Integration Framework software kits from the Global Technical Access Center (GTAC) and place them in the Deployment Center software repository.

   Software kits supporting several configurations are available from GTAC. 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 the **Deploy Software** tab. In the **Software** task, add the Teamcenter Integration Framework software to the **Selected Software** list.

5. In the **Options** task, select the options for your environment.

6. In the **Applications** task, 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. In the **Components** task, 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. In the **Deploy** task, generate the installation scripts.

9. Follow the steps in *Run the deployment scripts* in the Deployment Center help collection 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.

   Alternatively, you can start TEM from the Teamcenter software distribution image with the Java runtime environment (JRE) you want to use, for example

   ```
   TC_ROOT\install\tem -jre "C:\Program Files\Java\jdk1.7.0_17\jre"
   ```

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: 8090**
    **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 and UNIX systems, you can start Teamcenter Integration Framework in interactive mode by running the **trun** script in the **tcif/container/bin** directory. Alternatively, you can start the application as a background process by running the **start** script in the same 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.

    Alternatively, you can start TEM from the command line of the Teamcenter software distribution image with the Java runtime environment (JRE) that you want to use, for example:

    ```
    C:\apps\tc111_win\tem -jre "D:\Program Files\Java\jdk1.7.0_17\jre"
    ```

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, specify the location of the media used for the current Teamcenter installation for **Original Media Location**. Then, browse to and select the Teamcenter Integration Framework media location, adding it to the **Update Location** list.

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: 8090**
   **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 and UNIX systems, you can start Teamcenter Integration Framework in interactive mode by running the **trun** script in the **tcif/container/bin** directory. Alternatively, you can start the application as a background process by running the **start** script in the same 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 PLM 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.

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 help collection available on the Siemens PLM Software Doc Center.

1. From the Global Technical Access Center (GTAC), 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 from GTAC. 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 the **Deploy Software** tab. In the **Software** task, 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. In the **Options** task, review the existing options as needed.

6. In the **Applications** task, verify the Teamcenter Integration Framework applications are marked for updating.

7. In the **Components** task, 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. In the **Deploy** task, generate the installation scripts.

9. Follow the steps in *Run the deployment scripts* in the Deployment Center help collection 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. Custom feature files in the *TcIF_ROOT*/**container/deploy** directory that refer to the old Teamcenter Integration Framework version will not deploy correctly. Update the files to refer to the latest Teamcenter Integration Framework version.

13. 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.

14. Start Teamcenter Integration Framework. (On UNIX, 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.

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. Custom feature files in the *TcIF_ROOT*/**container/deploy** directory that refer to the old Teamcenter Integration Framework version will not deploy correctly. Update the files to refer to the latest Teamcenter Integration Framework version.

5. 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.

6. 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, feature files, and a set of Teamcenter Integration Framework configuration files are migrated.

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. Custom feature files in the *TcIF_ROOT*/**container/deploy** directory that refer to the old Teamcenter Integration Framework version will not deploy correctly. Update the files to refer to the latest Teamcenter Integration Framework version.

5. 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.

6. 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.

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. Custom feature files in the *TcIF_ROOT*/**container/deploy** directory that refer to the old Teamcenter Integration Framework version will not deploy correctly. Update the files to refer to the latest Teamcenter Integration Framework version.

e. 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

### 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 the migration, the migration stops and a message similar to **TcIF local datastore migration - abort** or **TcIF cluster datastore migration - abort** 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.

After a datastore migration successfully completes, it cannot be rerun.

## Update custom bundles

When upgrading Teamcenter Integration Framework, you may need to manually update custom bundles you are using. Update your 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.
- Ensure all feature files reference the current release version. Feature files that reference older versions will not deploy correctly.

## Retain business object definitions

The database schema business object definition (BOD) has changed for Teamcenter Integration Framework 10.1.5 later versions. This requires some manual 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 to a disk location 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

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 the "TcIF successfully started" message appears.

## Run Teamcenter Integration Framework as a service

You can run Teamcenter Integration Framework as a service on Windows systems. Depending on your installation, you may need to create the **\data\log** directory structure under your *TcIF_ROOT***tcif\container** directory so the log file is created in the correct location.

1. Start Teamcenter Integration Framework in a command shell using the **trun.bat** script in the *TcIF_ROOT***tcif\container\bin** directory.

2. Type the following commands in the Teamcenter Integration Framework command window:

```
karaf@TcIB> feature:install wrapper
karaf@TcIB> wrapper:install
```

3. Open the **karaf-wrapper.conf** file in the *TcIF_ROOT***tcif\container\etc** directory and add the following new parameters:

```
Update the following existing parameters:
wrapper.java.initmemory=512
wrapper.java.maxmemory=1024
```

   You can also rename the Windows service in this file.

4. Stop the running instance of Teamcenter Integration Framework by closing the command window.

5. Open a new command window and type the following command to install the service:

```
%TC_ROOT%\tcif\container\bin\karaf-service.bat install
```

6. Check the **wrapper.log** file in the *TcIF_ROOT***tcif\container\data\log** directory for any errors.

7. Confirm that the service is running by typing the following URL in a browser:

```
http://localhost:<web-ui-port>/tcif/controller/webclient
```

   <*web-ui-port*> is "8090" by default.

# 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. 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.

### 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.

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.

# 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 feature files and 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 PLM 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 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 only LDAP and single sign-on (SSO). An LDAP server is bundled with the Teamcenter Integration Framework framework and is the standard security system.

- 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 Spring service in Teamcenter Integration Framework.

- The Teamcenter Integration Framework user interface is extended to improve interaction with the messaging system. A dashboard shows the most recent changes.

- 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.

- 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**. 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.

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 contains an embedded LDAP server to store security credentials that you configure through the configuration user interface.

You can migrate any changes that you make to the **SOASavedQueryMapping** and **DataMapper** 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 new version of the mapping engine that provides better performance and supplies updated versions of the OOTB control files. You must update custom mapping control files to meet the requirements of the new mapper engine prior to migrating them.

## Update custom mapping control files

1. Replace all occurrences of the deprecated method of setting the **attrName** and **refElement** attributes to set a reference subelement on a factor to use the appropriate values for the **attrName**, **refElementKey**, and **split_token** attributes. Reference subelements with missing **refElementKey** attributes generate a warning.
2. Ensure all **GMSCrossFactorSetAttrs** elements contain **attrName**, **targetObject**, and **attrValues** attributes. If any are missing the mapping engine throws a **MappperException** error.
3. Any factor the specifies a parent factor (primary element in a child factor) must match an object in its parent factors. When no match is found, a warning is generated.
4. Add **postProcessorChain.#** values to the **mappingconfig.xml** file when multiple postprocessor routines are specified to indicate the order that they are invoked. This is required because the order of invocation is no longer hard-coded into the **mapperengine.jar** file. For example, if your mapping control file contains the following element:

   ```
   <param name="GMSCrossFactorSetAttrs" value="True"/>
   ```

Replace it with elements similar to the following:

```
<param
   name="postProcessorChain.1"
   value="com.teamcenter.datamapper.mapperengine.postprocessors.SimpleLookupTranslator;
com.teamcenter.datamapper.mapperengine.postprocessors.ElemIdAssigner;
com.teamcenter.datamapper.mapperengine.postprocessors.DuplicateRemover;
com.teamcenter.datamapper.mapperengine.postprocessors.PrefixRemover;
com.teamcenter.datamapper.mapperengine.postprocessors.ReferenceResolver;
com.teamcenter.datamapper.mapperengine.postprocessors.SetAttrsXFactor"/>
<param name="postProcessorChain.2"
   value="com.teamcenter.datamapper.mapperengine.postprocessors.SetAttrsXFactor;
com.teamcenter.datamapper.mapperengine.postprocessors.RemoveAndDedangler"/>
<param name="postProcessorChain.3"
   value="com.teamcenter.datamapper.mapperengine.postprocessors.RemoveAndDedangler"/>
```

- All postmapping control file routines (for example, **ElemIdAssigner**, **DuplicateRemover**, **PrefixRemover**, and **ReferenceResolver**) must be listed in the **postProcessorChain** variables.

- Some routines (for example, **SetAttrsXFactor** and **RemoveAndDedangler**) are listed twice. This is required because the routine executes per factor and also executes after all the data is gathered in the first pass.

- The **RemoveAndDedangler** routine replaces the **RemoveFlaggedObj.xsl** and **RemoveFlaggedRel.xsl** style sheets that were processed outside the mapping engine. These files are no longer necessary.

5. Add any required additional postprocessor routines as follows:

   a. Add Java classes to the **mapperengine2.jar** file or to a separate JAR file.

   b. Add the JAR file to the **CLASSPATH** variable and add its fully qualified name to one of the **postProcessorChain** variables in the **mappingconfig.xml** file.

   c. Extend the **SnippetPostProcessor** class and, at a minimum, override the **process()** method.

      The other methods in the **SnippetPostProcessor** class, **initialize()**, **initParams()**, and **finish()**, are available for customization but are not required.

      The following is an example of the syntax you use to specify custom postprocessor configuration variables:

      ```
      com.teamcenter.datamapper.mapperengine.postprocessors.SortByIsland
      {maxIslandFileHandles=250000,islandDivFactor=100}
      ```

      You must also write code in the **initParams()** method to process the **paramStr** argument.

6. Update any references postprocessor routines to the new name as follows:

| Original name | New name |
|---|---|
| **GMSCrossFactorSetAttrs** | **SetAttrsXFactor** |
| **HeaderHandler** | **HeaderCreator** |
| **PSAlternateListHandler** | **PSAListProcessor** |
| **AssmStrcQtyFixRequired** | **PSOccurrenceQtyProcessor** |
| **SharedMultiOccurrenceHandler** | **EffectivityIdProcessor** |

## 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.datamapper**

This bundle contains the mapping engine. It provides control file mapping and postprocessing. It also contains mapping designer projects. It does not publish to blueprint.

**Exported packages**

**com.teamcenter.datamapper.config**
**com.teamcenter.datamapper.mapperengine**

**Converted packages**

**mapperengine.jar**
**com.teamcenter.datamapper.***

> **Note**
>
> The existing package space remains the same because these classes are not in the **com.teamcenter.globalservices** package space. For example, the **com.teamcenter.datamapper** package space is the same in both the Global Services component and the Teamcenter Integration Framework bundle.

**com.teamcenter.esb.activemq**

This bundle wraps and deploys the Java Message Service (JMS) in the framework.

There are no exported or converted packages in this bundle.

**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**<br>**META-INF/MANIFEST.MF OSGI-INF\blueprint\bundle-context-osgi.xml**<br>**OSGI-INF\blueprint\bundle-context.xml**<br>**com\teamcenter\..*.class** |
| I18N Contents | **resources/***solutionName***.properties**<br>*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**<br>/*solution*/*solution-name*/**bos***BOSName***.xml**<br>/*solution*/*solution-name*/**config/***ConfigName***.xml** |

### Feature file

Your feature must contain is one feature XML file named **tcesb-***solution***-11.1.0-feature.xml**. The installed location must be *tcifInstallDir*/**tcif/container/deploy**.

### Bundle the solution kit

The bundled kit file must be a ZIP file containing the datastore contents, the solution OSGI bundles, and the feature file. The Ant build depends on the OSGI bundle module and the datastore module.

| | |
|---|---|
| Name | **TcIF***SolutionName***Bundle.zip** |
| Contents | **tcif/container/autoinstall/***Solution***DataStore.zip**<br>**tcif/container/bundles/com.teamcenter.***solution***.solution-***version***.jar**<br>**tcif/container/deploy/tcesb-***solution***-11.1.0-feature.xml** |
| Build output directory | **out/jars** |

### Include the bundled solution in the kit

The bundled solution must be available to the kitting process.

| | |
|---|---|
| Kitting File | **src/build/kits/kit_kitTcIntegrationFramework.xml** |
| Content to Add | Follow other examples in the file to add the **TcIF***Solution***Bundle.zip** reference to the kit. |

### Create the feature file

| | |
|---|---|
| Name | **feature_tcif***solution***intg.xml** |

### Add user displayed test to the language file for localization

| | |
|---|---|
| Name | **IntegrationFrameworkBundle_en_US.xml** |
| Source location | **src/ESB/install/lang** |
| Required changes | Add any messages or text that display to the user to the file. Do not change any other language files. The localization team does all translations necessary. |

### 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-10.1.4.jar**
Contains service and solution classes for substance compliance. This replaces the reactor.

**\solutions\subsCmp\tcif\container\bundles\tcsoa-subscmpl-fragment-10.1.4.jar**
This fragment adds the required support to the OOTB SOA connector to allow it to call an extension.

**\solutions\subsCmp\tcif\container\deploy\tcesb-subscmpl-10.1.4-features.xml**
This is a deployment file provisioning the deployment into the container.

## 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=barl%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, there is a **ProcessService** interface that contains the **getMethod()** method that you must implement to return a **java.reflect.Method** object referring to the method that the service uses to process messages. The processing method must take a message object and a credentials object. You can look at the **com.teacenter.esb.publish.BasePublishEventProcessor.groovy** package in the datastore for an example.

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.

# Chapter 4: Configuring and managing Teamcenter Integration Framework operation

# Chapter 4: Configuring and managing Teamcenter Integration Framework operation

## Configuring Teamcenter Integration Framework

### Teamcenter Integration Framework configuration overview

Use the Teamcenter Integration Framework configuration interface to configure Teamcenter Integration Framework components using a web browser. The configuration interface allows you to set values in Teamcenter Integration Framework configuration files in the datastore and provides existing values from files in the datastore that you can view or modify.

#### Running the configuration interface

Open the configuration interface with a browser using a URL with 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 Web UI port (**8090** by default).

The Teamcenter Integration Framework configuration interface displays the following configuration options 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. |
| **Mapping** | Configure and manage mapping relationships between sites. |

**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 activity status, failed message information, and business object definitions (BODs). See Failed messages.

**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 Help collection.

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 are serialized and deserialized when you upload them to or download them from the datastore.

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.

- Use the **Download** pane to view the current directory structure.
- Click the folder icons to show the contents of the directory.
- 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

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

You can package a directory structure containing file contents in a compressed archive format (JAR or ZIP files) and upload the entire contents to the appropriate location in the datastore by checking **Check to unpack contents of jar/zip file** before you upload the file.

Any directories that do not exist are created when you upload the file. You can also create new directories by manually typing them in the **Datastore location for object** box. This feature is useful for Teamcenter Integration Framework customizations.

You can disable access to this pane as described in Control file uploading.

#### Remove files from the datastore

Use the **Remove** pane to navigate to files that are no longer needed by Teamcenter Integration Framework. Removing files helps control the size of the datastore database as it evolves.

### Properties

**Properties** lists the exposed property bundles (areas) used to configure general properties of Teamcenter Integration Framework. The **display** attribute determines whether properties are displayed for the area's configuration.

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 

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 an area's properties

1. In the list of property areas, locate the area to edit and click ⌄.
2. Next to the property area name, click 

. Enter a name for the new property and click **OK**.
3. Click **Save changes** to save the changes to the datastore.

#### Refresh the list of property areas

As you work with properties, click  to update the list of displayed areas with the current contents of 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  .
2. Fill in the fields defining the characteristics of the new user. Fields marked with * are required to create a new Teamcenter Integration Framework user.

| | |
|---|---|
| **Name** | Specifies a user name associated with Teamcenter Integration Framework. |
| **Password** | Specifies the password associated with the user. |
| **Confirm Password** | 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 already supplied and 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 Installation/Customization* in the Teamcenter Help collection.

**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 **/webclient** 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 Installation/Customization* in the Teamcenter Help collection.

## SSL

The **SSL** tab lets you configure Teamcenter Integration Framework for HTTPS communications. Doing so provides secure encrypted communications from Teamcenter Integration Framework.

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 PLM Software recommends that you change the default password value (**secret**) 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.<br><br>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 

.
2. Fill in the fields defining the characteristics of the new connection. Fields marked with * are required to create a new data source connection.

   | | |
   |---|---|
   | **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.

### Refresh the list of data source connections

As you work with data source connections, click  to update the list of connections.

## 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 

.
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 Teamcenter Enterprise, the site ID is defined when you create the local site object. This value is supplied by GTAC.<br><br>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 in Teamcenter Environment Manager (TEM) 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 connector 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.

### Refresh the list of sites

As you work with sites, click to update the list of sites.

## Mapping

When transferring information between sites, the format of the data may need to be adjusted before being imported into the target site. The data can be reformatted by XSLT transforms, Java transforms, or other means. **Mapping** lets you configure and manage mapping relationships between sites.

### Filter the list of site mappings

To refine the list of site mappings, select a site type (source or target) and begin entering a string of characters in . As you type, the list updates to show only the site mappings with that string of characters in their ID for that site type.

### Create a new mapping

1. Click New.
2. Select the source and target sites from the list of available site configurations.
3. Create a sequence of mapper transforms by clicking and dragging transforms from the list of available transforms to the list of selected transforms. Click and drag the selected transforms to reorder them.
4. Click **Create** to create the mapping.

### Delete a site mapping

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

### Edit a site mapping

1. Locate the site mapping you wish to edit in the list of mappings.

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

3. Edit the mapping transforms. Click **Save** when complete to save the changes.

#### Refresh the list of site mappings

As you work with site mappings, click  to update the list of mappings.

## 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.

#### 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

### Failed messages

Teamcenter Integration Framework displays failed messages for the activities that it attempts. Click on a message in the **Data Views** pane to view its related details.

### 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 8 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.

## 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
```

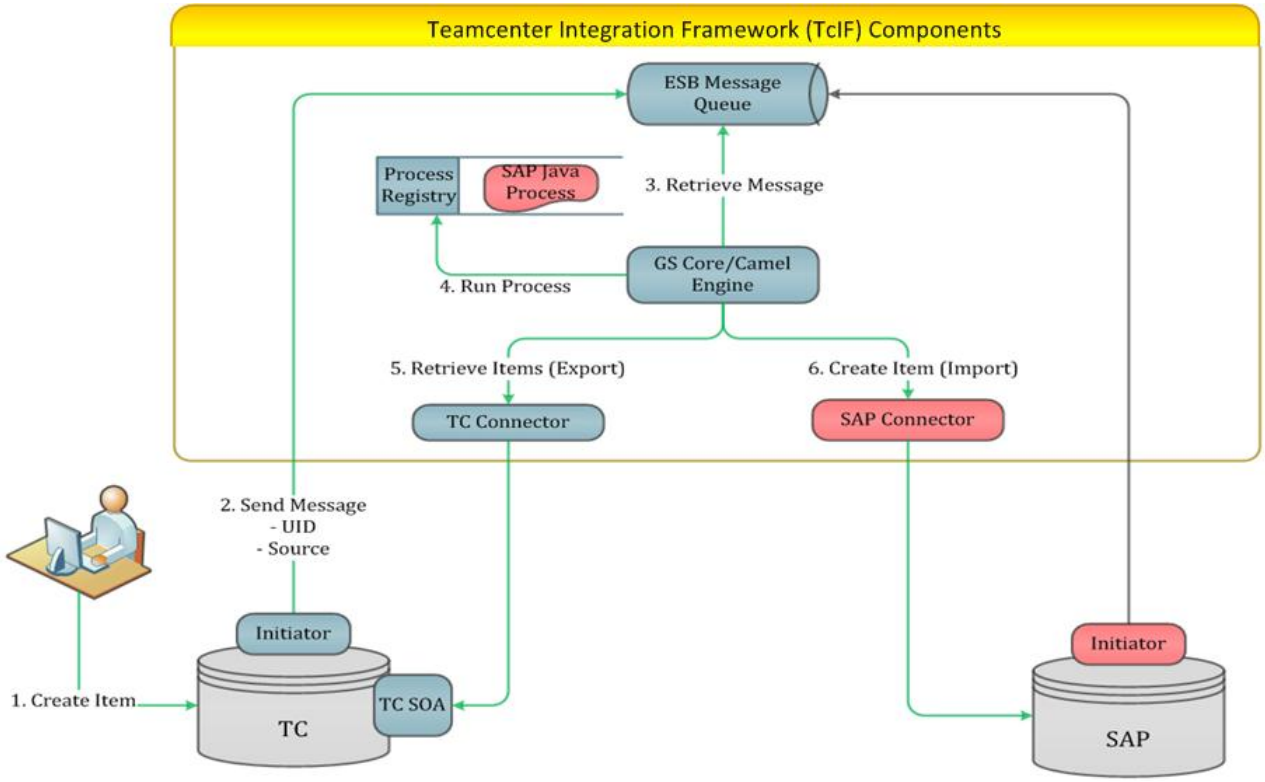
The email service configuration requires that you define an SMTP server and defaults for the message, such as a sending email account and possibly an email account to copy on all messages.

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.

For Teamcenter Integration Framework, the main difference is the client that calls Teamcenter Integration Framework. The Teamcenter Integration Framework is neutral; it executes a script process for data transfer that contains a set of steps or activities.

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:

| | |
|---|---|
| **Null** | Activity is not yet started or inactive. |
| **In Progress** | Activity has started. **Details** indicates **Active**. |
| **Waiting** | An error occurred during the activity. The process is waiting for a manual retry. After resolving the error condition, the user must use the **Resume** button on the **Detailed Transfer Status** dialog box to start the retry. |
| **Complete** | Activity has finished. **Details** indicates **Succeeded** or **Failed**. |
| **Aborted** | The transfer was aborted by a user prior to or during the activity. |

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.

Use the following process to stop Teamcenter Integration Framework

1. Navigate to the command window (Windows) or Karaf console (UNIX) in which Teamcenter Integration Framework is running.

2. Type **ctrl-d** or the Karaf command **system:shutdown**.

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

## Managing passwords

### Configure the Apache Karaf password

The Teamcenter Integration Framework integration server uses Apache Karaf to manage Open Service Gateway initiative (OSGi) bundles. There is a bundle management user interface (UI) component that is exposed through a URL endpoint. The default location for this endpoint is **http://*[host-name]*:8090/system/console**.

By default, the Karaf web console is disabled. To enable the web console, type the following command in the Karaf shell:

```
features:install webconsole
```

You can also add **webconsole** to following entry in **container/etc/or.apache.karaf.features.cfg** file to start the web console when Karaf starts:

```
featuresBoot=webconsole
```

You can configure Karaf passwords using the **users.properties** file located in the **tcif\container\etc** directory of your integration server installation location. This file contains the authorized users and the associated passwords for them in the following format:

```
user-name=password-value[,role][,role]...
```

Karaf passwords are used to access the SSH console, JMX management layer, and the web console, all using JAAS-based security authentication.

The initial configuration provides the **IFAdmin** user with **admin** (both case-sensitive) as the password and **admin** as the role. For security reasons, you must change the user and password values and enable password encryption, before you use the Teamcenter Integration Framework in a production environment.

For more about Karaf roles and realms and how to manage them, see the Karaf documentation provided at **karaf.apache.org.**.

## Enable Karaf password encryption

By default, Karaf passwords are stored in plain text in the **users.properties** file. For security, you must enable password encryption which replaces the plain text password with an encrypted version in the **users.properties** file the first time a user logs on to Karaf. Encrypted passwords are easily identified in the file by the **{CRYPT}** string in front of them.

To enable password encryption, enter the following commands in the Karaf console window:

```
# edit config
config:edit org.apache.karaf.jaas
config:propset encryption.enabled true
config:update
# force a restart
dev:restart
```

You must restart the Karaf console for the encryption feature to take effect.

## Change the Teamcenter Integration Framework security repository password

In addition to a single Teamcenter Integration Framework administrative account, the Teamcenter Integration Framework security repository has an administrative account defined as **uid=admin**,**ou=system**. This account provides access emergency maintenance purposes only. Siemens PLM Software recommends you change the account's default password value (**secret**) after you start Teamcenter Integration Framework the first time. You can use any existing LDAP client to make this change. If you do not have a client, you can download the Apache Directory Studio for this purpose.

Download the client for Windows, UNIX, and Linux systems from:

https://directory.apache.org/

The following steps are an example of how you change the password using the Apache Directory Studio.

1. In Apache Directory Studio, click **New Connection**  .
2. In the indicated box:
   - **Connection name**

     Type a descriptive name for the LDAP connection.
   - **Hostname**

     Type the name of your Teamcenter Integration Framework host.
   - **Port**

Type the Teamcenter Integration Framework LDAP port number (**14389** by default).

- **Encryption method**

  Leave the default **No encryption** value.

- **Provider**

  Leave the default **Apache Directory LDAP Client API** value.

3. Click **Check Network Parameter** to verify connectivity. If the connection is successful, click **Next**.
4. In the indicated box:
   - **Authentication Method**

     Leave the default **Simple Authentication** value.

   - **Bind DN or user**

     Type **uid=admin, ou=system**.

   - **Bind password**

     Type **secret**.
5. Click **Check Authentication** to verify you entered the proper credentials. If authentication is successful, click **Finish**.
6. In the left navigation pane, expand **ou=system** and select **ui=admin**. In the right pane, double-click **userPassword**.
7. Select the **New Password** tab and type your desired password in the **Enter New Password** box. Leave the other values as they are and click **OK**.

# 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. Uniquely identified by the **MessageID** value.

- **ActivityStatus**

  The **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 attempt to process a message. It is uniquely identified by the **ProcessID** value. A message is normally only processed once; however, if there are retries attempted, there can be multiple **ProcessStatus** objects associated with a **LogMessage** object. A process is composed of steps and possibly subprocesses. For instance export, mapping, and import are steps and 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 and has 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 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 **/etc/org.ops4j.pax.logging.cfg** XML-based configuration 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/container/etc/org.ops4j.pax.logging.cfg** 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.category.com.teamcenter.esb=TRACE
# Logging level for the TcGS JAXB classes
log4j.category.com.teamcenter.globalservices=TRACE
```

3. Stop the Teamcenter Integration Framework server, remove the log files, and restart the server.

You get full message tracing in the new log files until you change the configuration file and restart the Teamcenter Integration Framework server.

# Chapter 5: Customizing Teamcenter Integration Framework

# Chapter 5: Customizing Teamcenter Integration Framework

## Using Groovy scripts

### 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

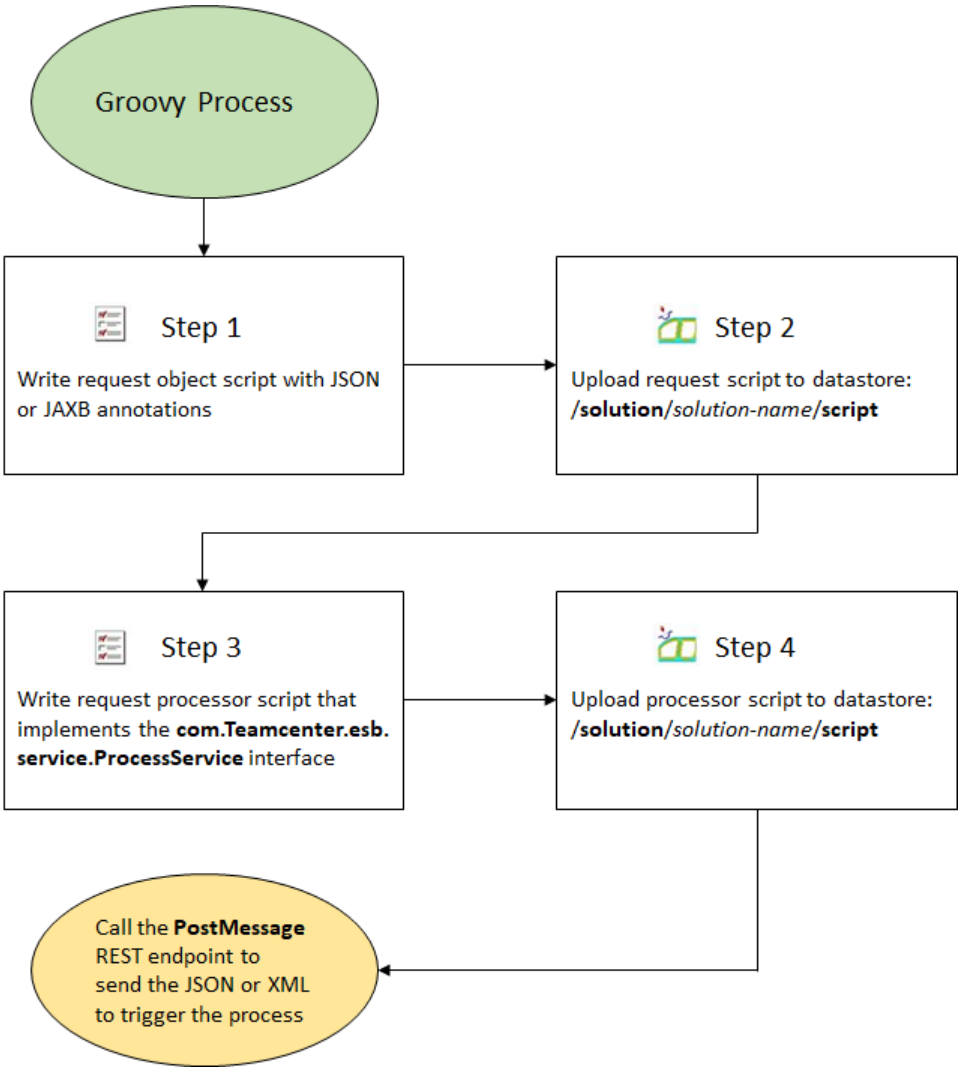
Scripts can start classes that are declared in other scripts. A Groovy classloader limitation exists where a Groovy class that is called from another Groovy class is not necessarily the latest version uploaded to the datastore. The Groovy classloader does not check to verify the version in the datastore 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.

## 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.

> **Note**
>
> Siemens PLM Software recommends that you use JSON Groovy objects and the **PostMessage** REST endpoint rather than SOAP and XML. Support for JAXB may be deprecated in a future release.

With JSON requests, the **PostMessage** endpoint takes a **service.messaging.ProcessRequest** JSON string in the body of the message. For example:

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

**synchronous** and **priority** are optional, and their default values are **true** and **4** respectively. Likewise, **destination** is optional. If no destination is specified, a rules engine is used to determine to which queue the message is sent.

Queues can have associated properties. If the message has properties that match those of a particular queue, that queue will be 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.)

For SOAP requests, you can send a web service request with a SOAP body containing XML that corresponds to a Groovy JAXB class in the datastore. For the **ProcessMessage**, the *message* JSON corresponds to a Groovy JSON or JAXB class in the datastore. Groovy JSON classes must extend the class **com.teamcenter.esb.internal.bind.JSONObject**. JAXB classes must extend the class **com.teamcenter.esb.internal.bind.JAXBBaseObject**.

XML namespaces should map onto the Groovy JAXB class package names. Tag names should map 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 JSON objects will have an **@class** field which maps onto a datastore location.

> **Note**
>
> Siemens PLM Software recommends that you put the **/script** location under the **/solution/***solution-name* location in the datastore.

Following is an example of a 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;
    }
}
```

This 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.

Teamcenter Integration Framework appends **Processor** to an input received in a format similar to **com.teamcenter.esb.example.ExampleRequest** through a SOAP request. It then looks for a Groovy script with that name (**com.teamcenter.esb.example.ExampleRequestProcessor**) to process the request. The processor script must implement the **ProcessService** interface and the

processing method must accept an argument whose type matches the request class. You upload the processor script into the datastore within the same solution as the request script and in the same directory (package) within the **/script** location.

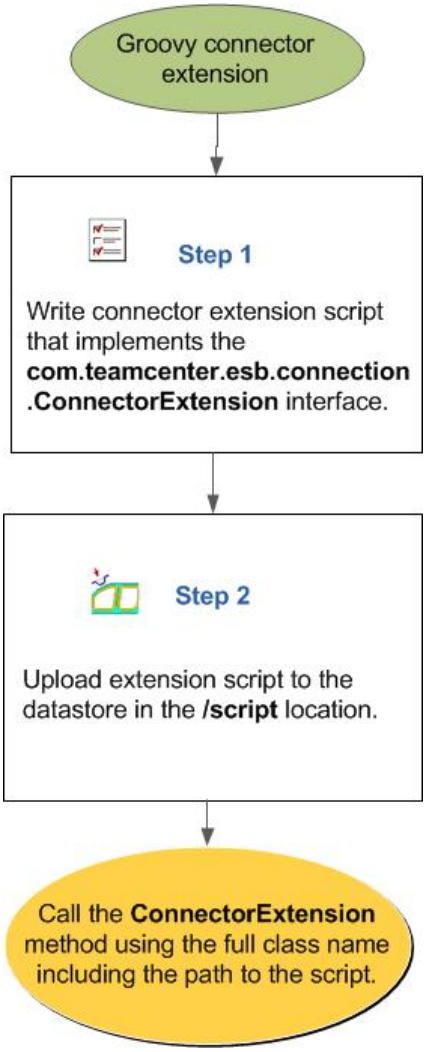
The following is an example of a processor script:

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

import com.teamcenter.esb.client.BOSClient;
import com.teamcenter.esb.client.BOSClientFactory;
import com.teamcenter.esb.commons.data.SerializableArrayList;
import com.teamcenter.esb.commons.data.SerializableList;
import com.teamcenter.esb.exception.ESBException;
import com.teamcenter.esb.model.security.Credentials;
import com.teamcenter.esb.service.ProcessService;

import java.lang.reflect.Method;

public class ExampleRequestProcessor implements ProcessService {

   public ExampleResponse process(ExampleRequest request, Credentials credentials) throws ESBException {
      ProxyServiceRequest serviceRequest = new ProxyServiceRequest();
      serviceRequest.setSiteId("111");  // should match some site in TcIF
      serviceRequest.setMethodName("com.tcesb.test.connector.extension.ComputeExtension");
      serviceRequest.getArguments().addAll(Arrays.asList("AVG", 9, 6, 3));
      Object result = ProxyService.callService(serviceRequest, credentials);
      ExampleResponse response = new ExampleResponse();
      response.setValue("response: " + result);
      return response;
   }

   @Override
   public Method getMethod()
      throws NoSuchMethodException {
      return getClass().getMethod("process", ExampleRequest.class, Credentials.class);
   }
}
```

The **com.teamcenter.esb.service.ProcessService** interface contains one **getMethod()** method that returns a **java.lang.reflect.Method** object. The method takes the given input and calls a **Credentials** object.

Processor scripts must use the **ProxyService** request to send execute requests to the connectors, where connector specific logic is executed, using the classpath of the connector.

## 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* 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 and topics) 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/rest/login**.
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 milliseconds, before an unprocessed job is moved to the stuck state.

   **Hung Time Out**

   Specifies the length of time, in milliseconds, 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. 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

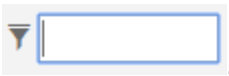
As a Teamcenter Integration Framework user or administrator, you can monitor Teamcenter Integration Framework queues.

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.

## 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

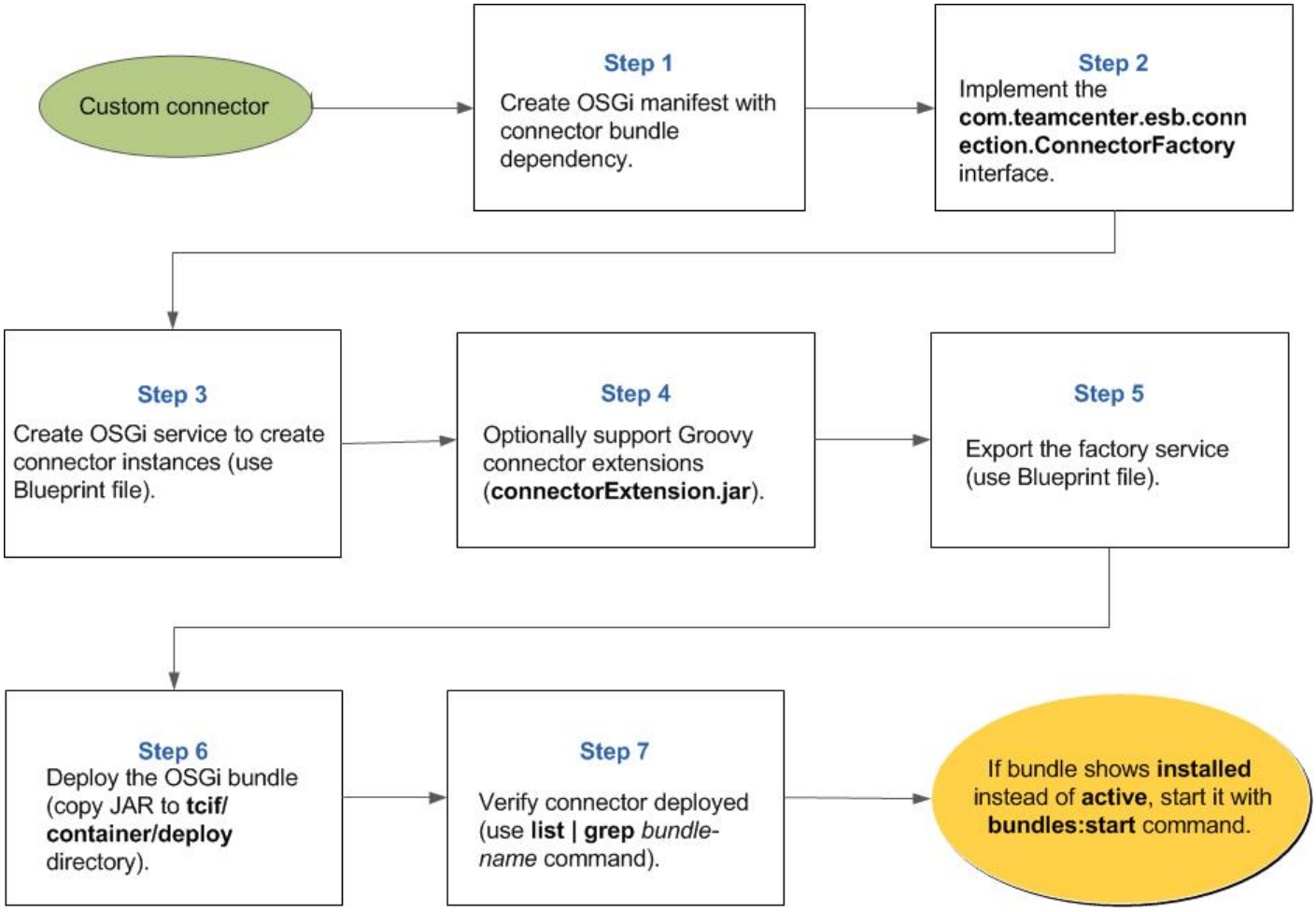
Teamcenter Integration Framework is based on an OSGi container with OSGi bundles. The bundles use the Blueprint dependency injection framework to provide configuration properties for the OSGi beans. These properties can be set in the **etc** directory of the container (**tcif/container/etc** directory)

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.***property-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.

# 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 | 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*

For more information on business object definitions and custom connectors, see *Platform Extensibility Configuration* in the Teamcenter help collection. In particular, see the section titled *Global Services connectors*.

## Extend a connector in Teamcenter Integration Framework

In Teamcenter Integration Framework, avoid creating subclasses from the connector implementation classes. Siemens PLM 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.

## 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 and replaces the **AuditService** class that used JAXB **AuditRequest** objects.

> **Note**
>
> The **AuditService** will be deprecated and should no longer be used.

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.setActivityCompleted(...)
auditService.audit(auditRequest); // audit end of 1st activity
```

4.

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

5.

```
ProcessUtil.setActivityCompleted(...)
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
```

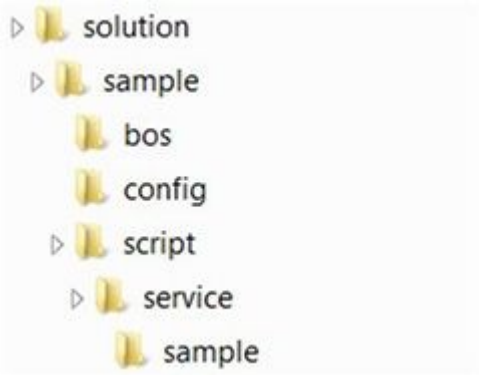
The auditing of the process as completed can be done in conjunction with the auditing call that marks the last activity as completed. That is, using only one call to **auditService.audit()** after marking both the **processInfo** completed and the **activityInfo** completed.

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 in the datastore configuration page.

## 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.

# Index

**Siemens Industry Software**

**Headquarters**
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 972 987 3000

**Americas**
Granite Park One
5800 Granite Parkway
Suite 600
Plano, TX 75024
USA
+1 314 264 8499

**Europe**
Stephenson House
Sir William Siemens Square
Frimley, Camberley
Surrey, GU16 8QD
+44 (0) 1276 413200

**Asia-Pacific**
Suites 4301-4302, 43/F
AIA Kowloon Tower, Landmark East
100 How Ming Street
Kwun Tong, Kowloon
Hong Kong
+852 2230 3308

**About Siemens PLM Software**

Siemens PLM Software, a business unit of the Siemens Industry Automation Division, is a leading global provider of product lifecycle management (PLM) software and services with 7 million licensed seats and 71,000 customers worldwide. Headquartered in Plano, Texas, Siemens PLM Software works collaboratively with companies to deliver open solutions that help them turn more ideas into successful products. For more information on Siemens PLM Software products and services, visit www.siemens.com/plm.