TEAMCENTER

Dispatcher — Deployment and Administration

Teamcenter 2412

SIEMENS

About Siemens Digital Industries Software

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

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

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

Contents

Configure the Dispatcher server

Configure the Dispatcher client for default translators

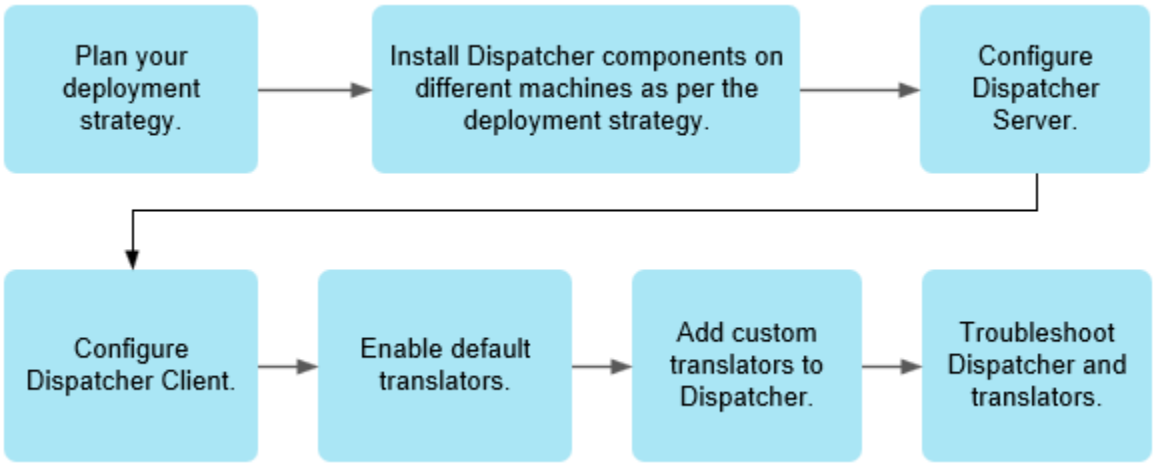
Enable default translators

Add custom translators to Dispatcher

1. Introduction to Dispatcher

Install and configure Dispatcher

A typical workflow for installing and configuring Dispatcher is as follows:



- **Plan your deployment strategy**
- **Install Dispatcher components on different machines as per the deployment strategy**
- **Configure Dispatcher Server**
- **Configure the Dispatcher client**
- **Enable default translators**
- **Add custom translators to Dispatcher**
- **Troubleshoot Dispatcher** or **troubleshoot translators**

Why use Dispatcher?

Dispatcher comprises the integration of two components: Dispatcher Server and Teamcenter. Dispatcher enables Teamcenter users to manage job distribution and execution.

With Dispatcher, you can:

- Asynchronously distribute jobs to machines with the resource capacity to execute the job. (The jobs can also be done in a synchronous manner on the local client machines, but this can introduce performance and administration issues based on the job load.)

- Reduce server load by distributing resource-intensive activities.
- Schedule high-CPU or high-memory jobs for off-hours processing.
- Get a grid technology solution to manage the job distribution, communication, execution, security, and error handling. You can scale and customize the Dispatcher system to meet specific site requirements.

Dispatcher is used to translate data files that are managed by Teamcenter to 3D or 2D file formats. You can:

- View, edit, and mark up data without requiring the original data's authoring tool.
- Share the data with other companies without distributing the original raw data (such as CAD and MS Office).
- Generate a lightweight data format known as visualization data. The visualization data can be attached along with the raw data and can be viewed by Teamcenter viewers or by third-party applications capable of reading and displaying the format.

Dispatcher components

- *Dispatcher Client*

  This component provides the framework to extract and load data to Teamcenter. It provides the communication mechanism to use the other Dispatcher Server components.

- *Scheduler*

  This component manages the distribution of translation requests across modules. Translation tasks are submitted to the scheduler by Teamcenter via RMI communication. The scheduler tracks which modules are available and the translators available with each module. It uses this information to distribute the translation tasks to keep each module running to its maximum capacity.

  The scheduler is a Java™-based RMI application. It comprises exposed interfaces used by the Dispatcher Client to send and receive information about Dispatcher requests. The scheduler communicates with the module using RMI.

- *Module*

  This component is used for sending tasks to specific translators required for the translation tasks. It provides the infrastructure for a common way to plug in and execute any translator and support the various command line options, parameters, and configuration files unique to each translator.

  The module is a Java-based RMI application. It has exposed RMI interfaces used by the scheduler to send and receive information about translation requests.

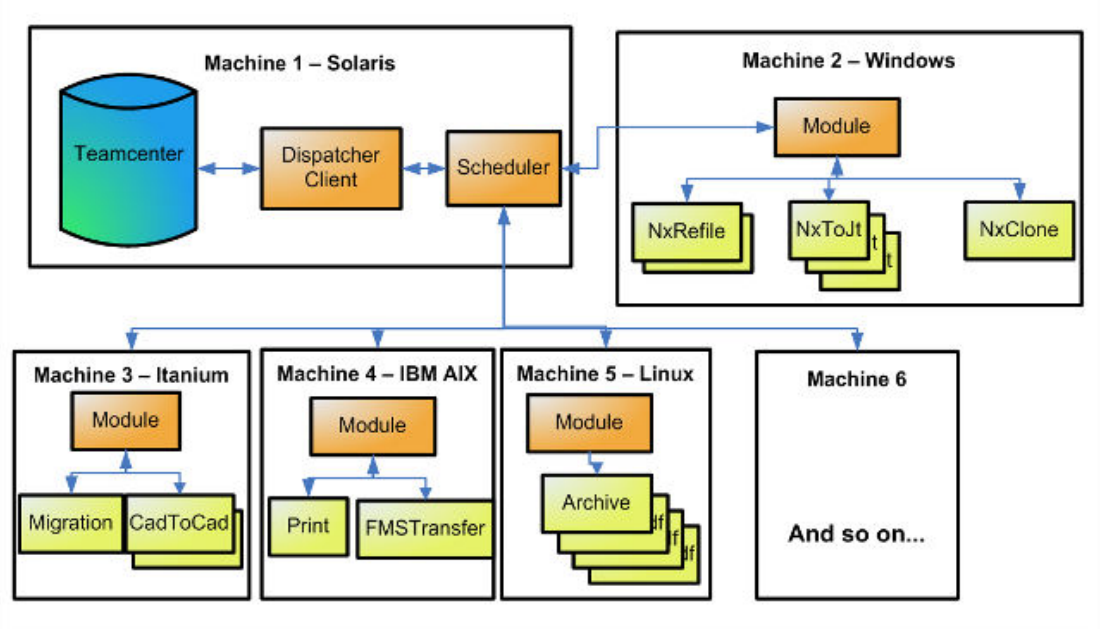
- *Translators*

  The translators used in the Dispatcher system are a combination of translators developed by Siemens Digital Industries Software and those developed by a third party. Translators can be driven by the Dispatcher system to use specific configuration files or parameters to support translation-specific behavior.

- *Dispatcher request administration console*

  This component manages and monitors the status of a task submitted for translation. You can use the Dispatcher request administration console in the rich client to:

  - View all translation requests.
  - Resubmit a translation request for processing.
  - Delete a translation request.



2. Dispatcher deployment considerations

Selecting hardware for Dispatcher Server components

Selecting hardware for the module and translators

The module is where all the intense computational process of translations occur. It must be sized sufficiently to handle the expected translation throughput.

The module hardware requirement is governed by the translator hardware requirements, number of translators installed, and the number of maximum potential translation tasks (number of users simultaneously submitting tasks).

An enterprise class server machine is highly recommended for the module, as these machines have the ability to scale the number of CPUs and memory much more than a normal workstation.

Selecting hardware for the scheduler

The scheduler provides the communication from Teamcenter clients and manages the queue and job distribution to the modules.

The scheduler is only a pipeline mechanism to manage the translation requests used by modules. The scheduler requires minimum machine resource to run; however, it does hold an object in memory for each translation request that it is managing.

Selecting hardware for the dispatcher client

The dispatcher client sends the translation task from Teamcenter to Dispatcher Server. Only one dispatcher client is required per Teamcenter server installation.

The dispatcher client for Teamcenter manages the data extraction and data loading into Teamcenter. While the data extracting and loading are not computationally intensive, it does require a machine with communication pipeline to the server as well as enough disk space and memory needed for the operations. Although not required, the recommended approach is to have the Dispatcher client running on the same machine as the Teamcenter server.

Criteria for choosing hardware for Dispatcher Server components

Prior to determining the type of machines to use for translations, determine the Dispatcher Server system usage to set up the correct environment to support the expected throughput.

Number of users

The number of users submitting translation tasks must be accounted for when trying to get the maximum amount of throughput.

If a site has 100 users who can potentially submit 100 simultaneous translation tasks, and the expected throughput is 1 hour, there must be sufficient computational capacity to handle the load. This can be accomplished by providing enough CPU and memory to handle the 100 tasks. Increasing load capacity can also be handled by adding additional modules to handle the additional throughput. As throughput requirements increase or decrease, modules can be started or stopped dynamically to handle the load requirements.

Size and complexity of the translation models

Knowing the average size of the models that use the system gives an idea as to the minimal requirements required for supporting the normal day-to-day operational needs. Consider the size of the models to be submitted for translation estimates when planning for the number of translations that can be executing simultaneously.

> Note:
>
> If a machine has 10 GB memory available, and the average size of the model requires 4 GB of memory to translate, the maximum number of translators that should be configured to run at a given time should be two.

Model complexity influences translation times. Models with many curves are more geometrically complex to translate than models with many straight edges.

Computational ability of the translation machine

The components required to get the data from Teamcenter to the translators are mainly pipeline mechanisms for moving data and communication and require proper network bandwidth for the expected data load, which is unique to each site. The machine performing the actual translation task must have enough CPU, memory, and disk space capacity to load and translate the models.

Disk space requirements

Understanding the disk space requirements needed ensures that the translations do not run out of disk space during a translation. To obtain the required disk space requirements, multiply the average sized model with the number of possible simultaneous translations.

Average model size * number of translations = Average disk space needed at any given time

You also need to account for the generated data, logs, and CAD system files while calculating the disk space needed.

Planning the deployment of the dispatcher components

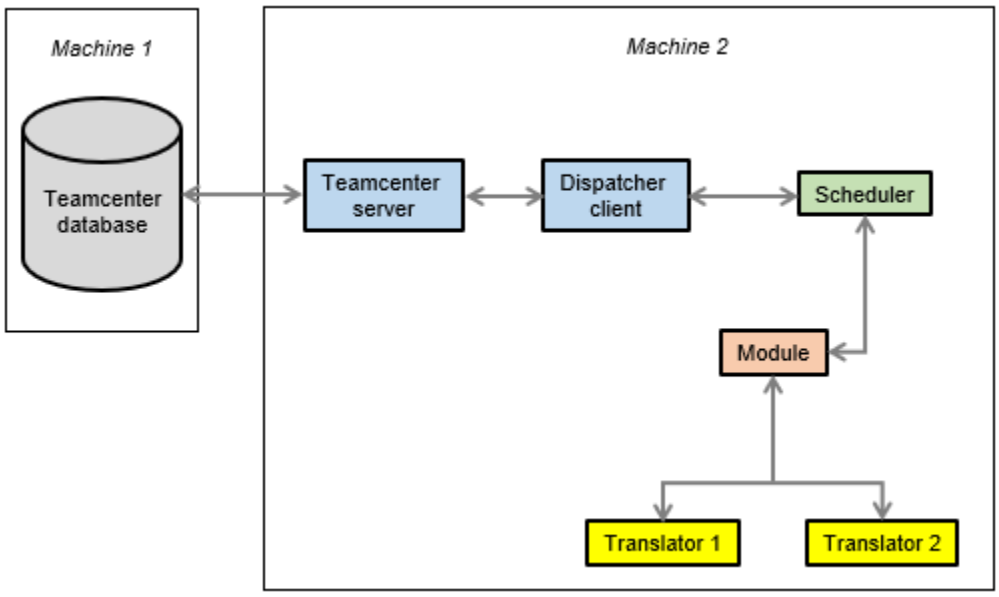
Many deployment strategies for Dispatcher Server components can be defined to meet specific needs. The main goal of defining a deployment layout is to have the Dispatcher Server components run on machines that have the resource capacity (CPU, memory, disk space, and network bandwidth) to handle the expected throughput of the site.

You can install Dispatcher Server components on the same machine. While this minimizes the administrative overhead, it reduces the scalability. A deployment across multiple machines allows distribution of the needed computing resources and provides scalability. However, distributing the services across machines increases administrative overhead.

The Dispatcher Server components can be configured to support various domain requirements. You can select from various deployment layouts based on your requirements.

All local (same host)

All Dispatcher Server components run on the same host. This deployment is typically used if the translation load on your site is less or if you have a high-end machine with a large amount of CPU, memory resources, and sufficient disk space capable of handling the expected translation throughput. A typical 2-tier installation is as follows.

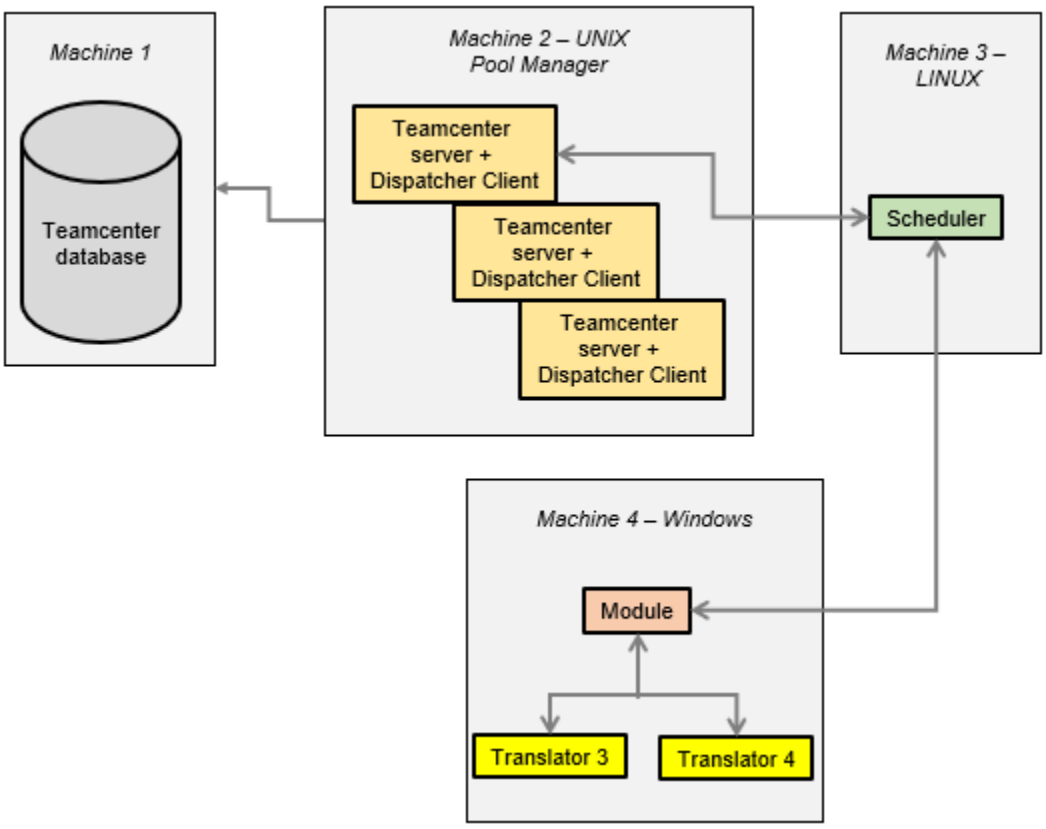


To install Dispatcher Client, run TEM, select **Corporate Server**, select **Teamcenter Foundation** (default for 2-tier) and **Dispatcher Client**, specify the Teamcenter data directory of the Teamcenter server, and the host name and the host port of the machine on which scheduler is installed.

Distributed on multiple hosts

A 4-tier installation with a separate module

This deployment strategy is to install the Teamcenter server and the Dispatcher client on the same machine and the scheduler and the module on separate machines.

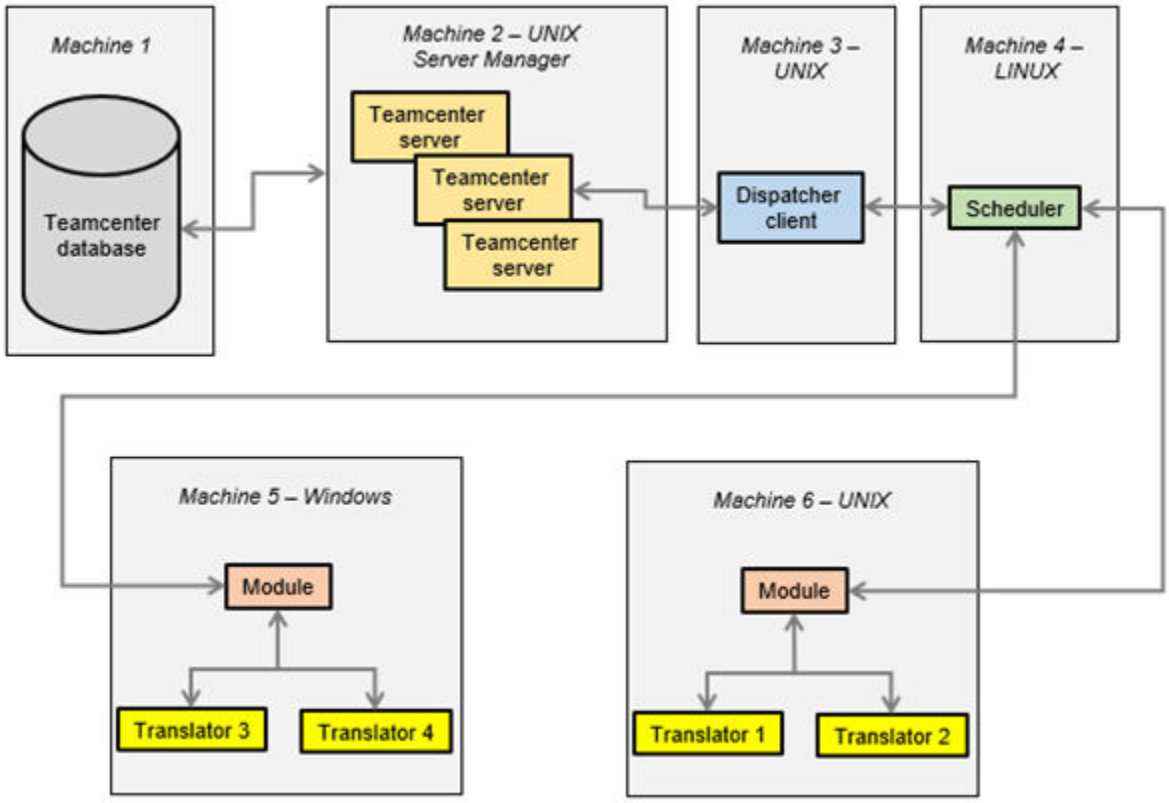


1. To install scheduler on *Machine 3*, run TEM on *Machine 3*, select **Dispatcher Server** and specify the Teamcenter root directory and the Teamcenter data directory of the Teamcenter server (that is, *Machine 2*).

2. To install the module on *Machine 4*, run TEM on *Machine 4*, select **Dispatcher Server**, and specify the staging directory (separate machine) and the host name and host port of the machine on which the scheduler is installed (that is, *Machine 3*).

A 4-tier installation with two separate modules

Here, Teamcenter server, the Dispatcher client, the scheduler, and modules are installed on separate machines.



1. To install the Dispatcher client on *Machine 3*, run TEM on *Machine 3*, select **Dispatcher Client** and select the **RMI** option, and specify the Dispatcher server host name and the Dispatcher server port (that is, *Machine 2*).

   In addition, specify the staging directory and the Dispatcher client proxy user name and password. The staging directory can be a separate machine.

2. To install scheduler on *Machine 4*, run TEM on *Machine 4*, select **Dispatcher Server**, and specify the Dispatcher client host name and host port (that is, *Machine 3*).

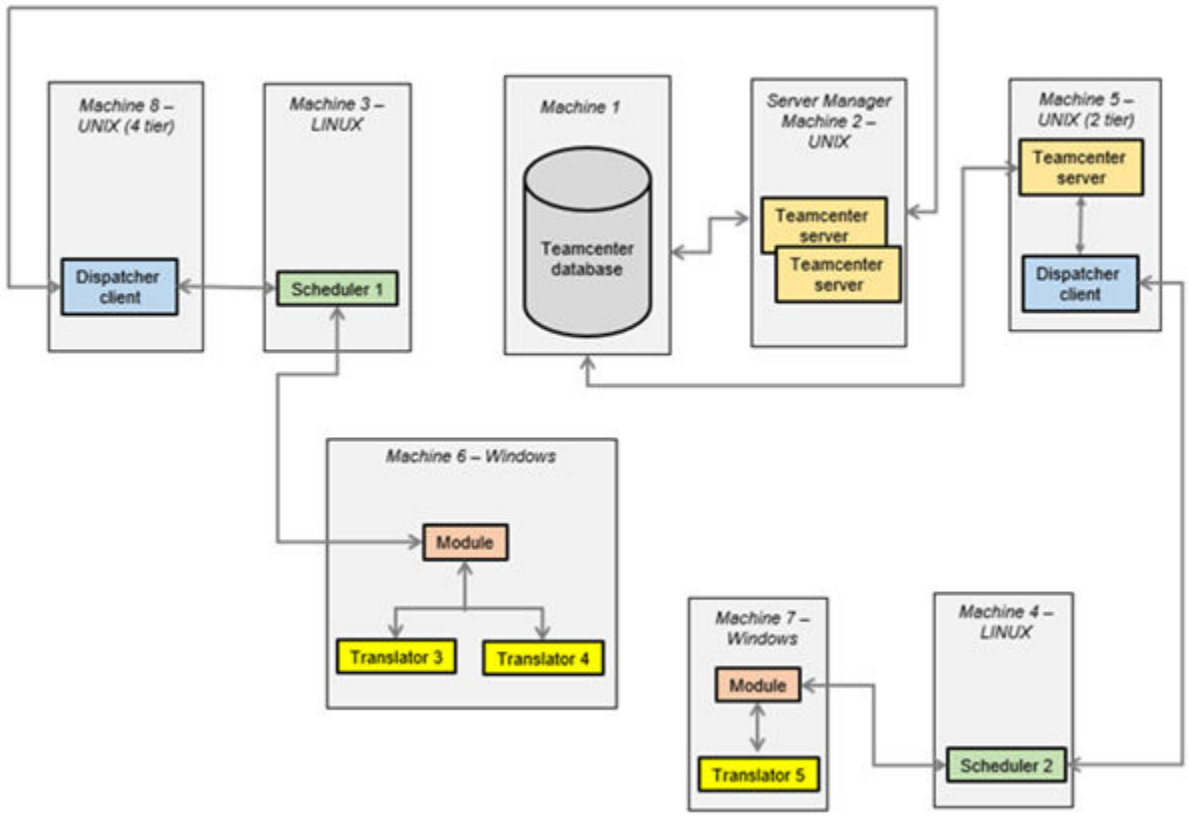
3. To install the module on *Machine 5*, run TEM on *Machine 5*, select **Dispatcher Server**, and specify the staging directory (separate machine) and the host name and host port of the machine on which scheduler is installed (that is, *Machine 4*).

A 4-tier installation with two separate modules where one module is for a high throughput, dedicated translator

There are two setups in this example.

- The first is a dedicated setup for a high throughput, dedicated translator. This has Teamcenter server and the Dispatcher client installed on the same machine, and scheduler and the module are installed on separate machines.

- The second is a setup for multiple translators on a single machine. This has Teamcenter server, the Dispatcher client, the scheduler, and the module installed on separate machines.



Plan the number of translations running on a module

You must first make a rough estimate of how many translator tasks are required per day. Then you must decide whether the machine can handle all these translators.

You can decide the number of translations that can run on a module by setting the following properties:

- **MaximumTasks** controls the maximum number of tasks allowed in the module at a particular instant. Use this property to control the maximum number of services or translations running on the machine. For more information, see *DispatcherRoot***Module\conf\transmodule.properties** file.

- **maxlimit** defines the maximum number of translations that can run simultaneously. Default is **1** even if you do not set this value. For more information, see *DispatcherRoot***Module\conf\translator.xml** file.

Example 1

You estimate that **10** translator tasks can be handled at the initial stage. This module has three services installed on it. They are as follows:

- **Translator1**

  Runs **5** translator tasks and they are the most important ones.

- **Translator2**

  Runs **2** translator tasks and they are based on license restrictions. Some translators have license restrictions and only two licenses of the software are available at a time.

- **Translator3**

  Runs **3** translator tasks.

In this example, edit the *DispatcherRoot***Module\conf\transmodule.properties** file to set the **MaximumTasks=10** property. This is based on the number of services you had estimated initially.

Then edit the *DispatcherRoot***Module\conf\translator.xml** file to specify values as follows:

```
<Translator1 isactive="true" maxlimit="5" provider="SIEMENS"
service="Translator1">
<Translator2 isactive="true" maxlimit="2" provider="SIEMENS"
service="Translator2">
<Translator3 isactive="true" maxlimit="3" provider="SIEMENS"
service="Translator3">
```

Example 2

After observing the performance of the module for some days, you want to increase the number of translations to **13** as follows.

- **Translator1**

  Runs **7** translator tasks as they are the most important ones.

- **Translator2**

  Runs **2** translator tasks and they are based on license restrictions. Some translators have license restrictions and only two licenses of the software are available at a time.

- **Translator3**

  Runs **4** translator tasks.

In this example, you edit the *DispatcherRoot***Module\conf\transmodule.properties** file to set the **MaximumTasks=13** property. This is based on the number of services you have decided after increasing the load.

Then edit the *DispatcherRoot***Module\conf\translator.xml** file to specify values as follows:

```
<Translator1 isactive="true" maxlimit="7" provider="SIEMENS"
service="Translator1">
<Translator2 isactive="true" maxlimit="2" provider="SIEMENS"
service="Translator2">
<Translator3 isactive="true" maxlimit="4" provider="SIEMENS"
service="Translator3">
```

This is a continuous process and you must iterate these steps to decide the maximum number of translations that can be run on a module.

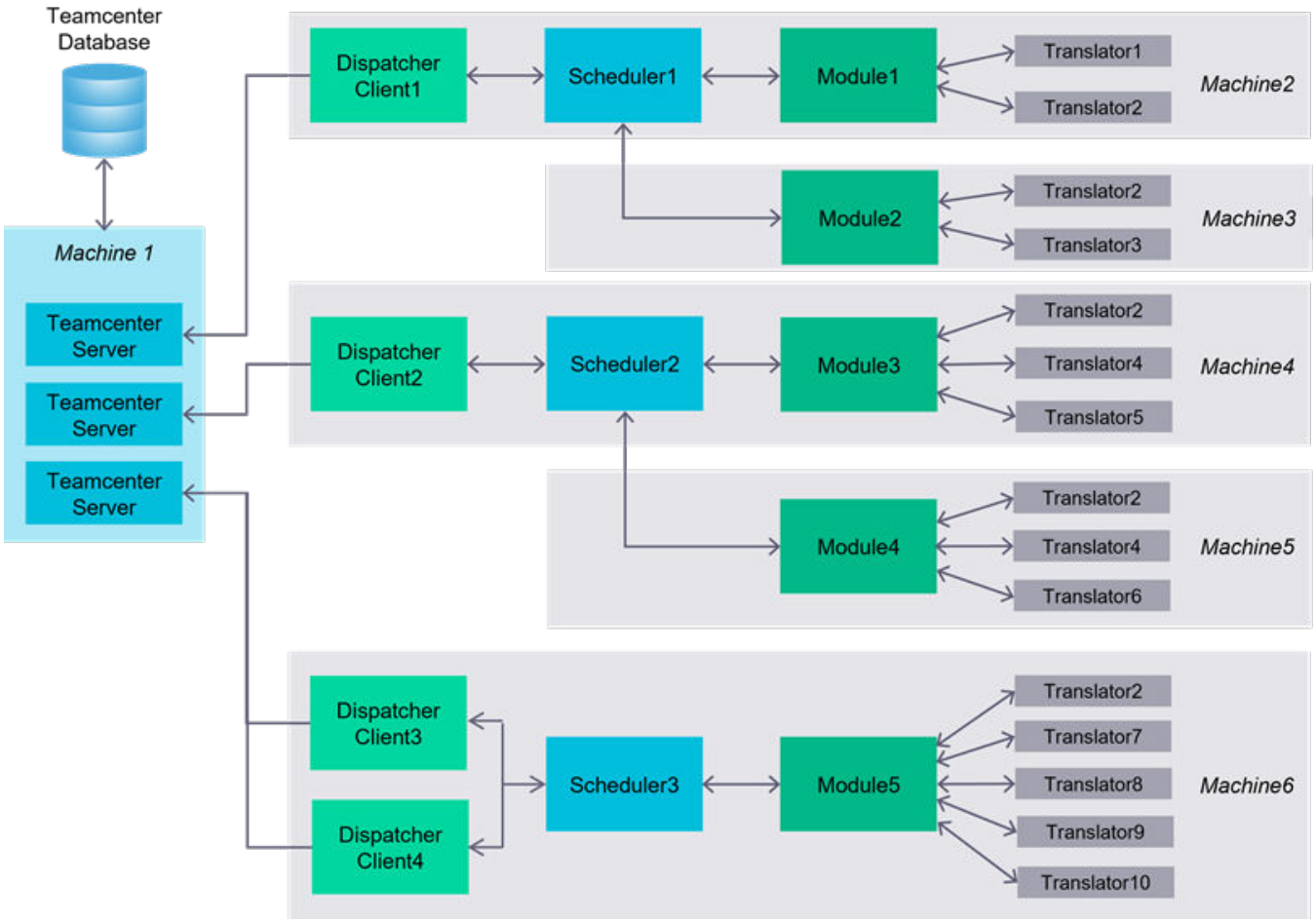
Manage Dispatcher requests using filters

You can manage Dispatcher requests using Dispatcher filters. The following are the main types of available filters. Other filter types are also supported. For more information, see the **Filters** section in *DispatcherRoot***DispatcherClient\conf\service.properties**.

Filter type	Description
TranslatorFilter	Filters out all Dispatcher requests without one of the specified provider or translator name pairs. Translator properties are specified as comma separated values.
PriorityFilter	The Dispatcher client processes all Dispatcher requests with the specified translators. Translator properties are specified as comma separated values of provider and translator name.
TranslationArgsFilter	Processes Dispatcher requests only if the specified translation arguments are available.

For information about customizing filters, see *What are filter classes?*

Example:



In this example, the **TranslatorFilter** is used. You can edit the **Filters** section in *DispatcherRoot***DispatcherClient\conf\service.properties** to specify these values.

- Example for **Dispatcher Client1** (**Machine2** and **Machine3**):

  ```
  Service.Filters=TranslatorFilter
  Service.TranslatorFilter.Provider=SIEMENS,SIEMENS,SIEMENS
  Service.TranslatorFilter.Translator=Translator1,Translator2,Translator3
  ```

  In this example, the **TranslatorFilter** is applied and only Dispatcher requests intended for the SIEMENS **Translator1**, SIEMENS **Translator2**, and SIEMENS **Translator3** services are processed for **Machine2** and **Machine3**. Both these machines share **Dispatcher Client1**.

- Example for **Dispatcher Client2** (**Machine4** and **Machine5**):

  ```
  Service.Filters=TranslatorFilter
  Service.TranslatorFilter.Provider=SIEMENS,SIEMENS,SIEMENS,SIEMENS
  Service.TranslatorFilter.Translator=Translator2,Translator4,Translator5,
  Translator6
  ```

In this example, the **TranslatorFilter** is applied and only Dispatcher requests intended for the SIEMENS **Translator2**, SIEMENS **Translator4**, SIEMENS **Translator5**, and SIEMENS **Translator6** services are processed for **Machine4** and **Machine5**. Both these machines share **Dispatcher Client2**.

- Example for **Dispatcher Client3** and **Dispatcher Client4** (**Machine6**):

  ```
  Service.Filters=TranslatorFilter
  Service.TranslatorFilter.Provider=SIEMENS,SIEMENS,SIEMENS,SIEMENS,SIEMENS
  Service.TranslatorFilter.Translator=Translator2,Translator7,Translator8,Translator
  9,Translator10
  ```

  In this example, the **TranslatorFilter** is applied and only Dispatcher requests intended for the SIEMENS **Translator2**, SIEMENS **Translator7**, SIEMENS **Translator8**, SIEMENS **Translator9**, and SIEMENS **Translator10** services are processed for **Machine6**.

How the Dispatcher failover mechanism works

- *Supports multiple Dispatcher installations*

  See the **diagram about managing dispatcher requests using filters**.

  The Dispatcher has the following services:

  1. Scheduler: Manages the distribution of Dispatcher requests across modules.
  2. Module: Manages translator execution.
  3. Dispatcher Client: Extracts and loads data to Teamcenter.

  - Multiple Dispatcher installations can support the same translators, for example, **AsyncService**, to handle failover.
  - All Dispatcher services support the **-ping** argument with every startup script to check if the services are running.

- *Failover for Dispatcher Client*

  - Each Dispatcher Client queries Teamcenter for specific translator type requests based on filter configurations.

    For more information, see **Manage Dispatcher requests using filters**.

  - The same request is not processed by multiple Dispatcher Clients.
  - If the Dispatcher Client goes down, other Dispatcher Installations keep processing the requests for a given type of translator.

  - Dispatcher Client supports retries if there are issues with the pool manager or **TcServer**.

- *Failover for Scheduler*

  - Scheduler has its own database to manage all the requests currently being processed.
  - If the Scheduler goes down, these requests are reprocessed when the scheduler is available.
  - You can clean the Scheduler cache directory and run the Dispatcher utility for the requests that need to be resubmitted.
  - Requests can be resubmitted by using **dispatcher_util** from the *TC_ROOT***bin** directory. This changes the state of the request to **INITIAL** and the next available Dispatcher installation processes the request.

- *Failover for Module*

  When a module goes down, the scheduler reroutes the requests being processed to the next available module.

  In the **diagram about managing dispatcher requests using filters**, when **Module1** or *Machine1* fails, **Scheduler1** directs the translation requests to **Module2** or *Machine2* running in the same Dispatcher instance. Similarly, when **Module2** or *Machine2* fails, **Scheduler1** directs the translation requests to **Module1** or *Machine1* running in the same Dispatcher instance.

- *Dispatcher Metrics*

  The Dispatcher writes a history log with a record of all the Dispatcher request state changes and the time of the state change. You can parse this log file to get the required metrics data.

3. Install and configure Dispatcher components

Install Dispatcher using TEM

Install Dispatcher components using TEM

Install the following components as per your deployment strategy:

- Install the scheduler on a separate machine that communicates with both the Dispatcher client and the modules.
- Install Dispatcher client on the same machine as the Teamcenter server or on a different machine that communicates with the server manager.
- Install modules on different machines depending on the translation load.
- Install **Dispatcher Client for rich client** on the servers where you install Teamcenter. This is for the Dispatcher user interface, that is, the translate menu and the right-click translate options in the rich client.
- (Optional) Install Admin Client on the server where you install Teamcenter. This is to verify the Dispatcher Server installation.

Install all Dispatcher components

> Note:
>
> The following instructions are for installing Dispatcher with a new installation of Teamcenter.

1. Start Teamcenter Environment Manager (TEM).

   If you are installing Dispatcher on an already existing Teamcenter server, launch TEM from your Teamcenter environment.

   If this is a first-time installation of a Teamcenter server and you want to include Dispatcher, run the **TEM.bat** (Windows) or **TEM.sh** (Linux) file from your installation source.

2. In the **Solutions** panel, select **Dispatcher (Dispatcher Server)**.

3. Perform the following steps in the **Select Features** panel:

   a. Under **Enterprise Knowledge Foundation**, select the following:

   - **Dispatcher Client for Rich Client**

     Installs the Dispatcher client on the rich client.

     This feature requires either the Teamcenter two-tier rich client or the Teamcenter four-tier rich client.

   - **Dispatcher Server**

     Installs Dispatcher Server components: scheduler, module, and Admin Client.

   - **Dispatcher Client (4-tier)**

     Installs the Dispatcher client in 4-tier mode.

     Siemens Digital Industries Software recommends the 4-tier mode as it requires a smaller footprint and is more scalable.

   - **Dispatcher Client (2-tier)**

     Installs the Dispatcher client in 2-tier mode.

     Siemens Digital Industries Software recommends the 4-tier mode.

     > Note:
     >
     > The Dispatcher Client (2-tier) is deprecated and will be removed in a future version of Teamcenter.

4. Enter information as needed in subsequent panels.

5. In the **Dispatcher Components** panel, do the following:

   a. In the **Dispatcher Root Directory** box, type or select the Dispatcher root directory.

   Keep the Dispatcher root directory as close to the Teamcenter root directory as possible.

   This directory is referred to as *DISP_ROOT*.

   b. Select the **Install Scheduler** check box to install the scheduler.

c. Select the **Install Module** check box to install the module.

If you select the **Install Module** check box, the **Staging Directory** box is activated.

In the **Staging Directory** box, you can choose the default staging directory or type a new one.

> Note:
>
> If you are installing the module and the scheduler on the same machine, the **Scheduler Host** and the **Scheduler Port** boxes are disabled.

d. Select the **Install Admin Client** check box to install the Admin Client.

e. Click **Next**.

6. In the **Dispatcher Settings** panel, do the following:

a. Select the logging level in the **Enter Logging Level** box.

The **Dispatcher Services Log Directory** is automatically populated with the location of the log directory.

b. Select the **Install Documentation** check box to install Javadocs for the Dispatcher components.

c. In the **Documentation Install Directory** box, type or browse to the location where you want the documentation installed.

d. If you want to start Dispatcher services after installation, select the **Starting Dispatcher Services** check box.

- To start Dispatcher services as a Windows service, click **Start Dispatcher Services as Windows Services**.
- To start Dispatcher services from the console, click **Start Dispatcher Services at console**.

e. Click **Next**.

7. In the **Select Translators** panel, select the translators you want to enable and click **Next**.

8. In the **Translator Settings** panel, provide configuration information for the translators you selected in the **Select Translators** panel and click **Next**.

9. In the **Dispatcher** panel, do the following:

a. In the **Dispatcher Server Hostname** box, type the name of the server where the translation server is hosted.

b. In the **Dispatcher Server Port** box, type the port to be used for the Dispatcher Server. The default port number is **2001**.

The scheduler port is used.

Make sure that the port you choose is not used by any other process.

c. In the **Staging Directory** box, type or browse to the location to be used as the staging directory for the Dispatcher client.

d. In the **Dispatcher Client Proxy User Name** box, type the name of the proxy user who uses Dispatcher services.

e. In the **Dispatcher Client Proxy Password** box, type the password for the proxy user.

f. In the **Dispatcher Client Proxy Confirm Password** box, type the password for the proxy user.

g. In the **Polling interval in seconds** box, type the time in seconds that the Dispatcher client should wait for before querying for Dispatcher requests.

h. In the **Do you want to store JT files in Source Volume?** box, select **Yes** if you want to store visualization files in the associated visualization dataset.

If you select **No**, the visualization files are stored in the *Dispatcher Client Proxy User* default volume.

i. Click **Next**.

10. In the **Dispatcher Client** panel, do the following:

a. Select the logging level in the **Enter Logging Level** list.

b. The **Dispatcher Client Log Directory** is automatically populated with the location of the log directory.

c. In the **Advanced Settings** section, do the following:

A. In the **Do you want to Update Existing Visualization Data?** box, select **Yes** if you want to update existing visualization data to the latest version.

B. In the **Deletion of successful translation in minutes** box, specify the time (in minutes) that the dispatcher client should wait before querying and deleting successful translation request objects.

If the interval is set to zero, the translation request cleanup is not performed.

C. In the **Threshold time in minutes for successful translation deletion** box, specify the time (in minutes) that must pass after a successful translation request object is last modified before it can be deleted.

D. In the **Deletion of unsuccessful translation in minutes** box, specify the time (in minutes) that the dispatcher client should wait before querying and deleting unsuccessful translation request objects.

E. In the **Threshold time in minutes for unsuccessful translation deletion** box, specify the time (in minutes) that must pass after an unsuccessful translation request object is last modified before it can be deleted.

d. Click **Next**.

11. In the **Confirm Selections** panel, click **Next**.

The installation starts.

Install Dispatcher while creating a new configuration

1. Start Teamcenter Environment Manager.
2. In the **Maintenance** panel, choose the **Configuration Manager** option and click **Next**.

   Teamcenter Environment Manager displays the **Configuration Maintenance** panel.
3. In the **Configuration Maintenance** panel, choose the **Add new configuration** option and click **Next**.

   Teamcenter Environment Manager displays the **New Configuration** panel.
4. Enter a description of and unique ID for the configuration you are creating and click **Next**.

   Teamcenter Environment Manager displays the **Solutions** panel.

   Ensure that you select the **Dispatcher (Dispatcher Server)** option.
5. Select the components to include in the configuration and click **Next**.

   Teamcenter Environment Manager displays the **Select Features** panel.
6. In the **Select Features** panel, select the Dispatcher components.

   The Dispatcher components are available under **Extensions→Enterprise Knowledge Foundation.**

From this point, the procedure is the same as for the first configuration in an installation. Teamcenter Environment Manager displays additional panels depending on the components you are including in the configuration.

Install Dispatcher while performing maintenance on an existing configuration

1. Start Teamcenter Environment Manager.
2. In the **Maintenance** panel, choose the **Configuration Manager** option and click **Next**.
3. In the **Configuration Maintenance** panel, choose the **Perform maintenance on an existing configuration** option and click **Next**.
4. From the list of configurations, select the configuration you want to modify and click **Next**.
5. In the **Feature Maintenance** panel, select **Add/Remove Features** and click **Next**.
6. In the **Select Features** panel, select the Dispatcher components.

   The Dispatcher components are available under **Extensions→Enterprise Knowledge Foundation.**

   From this point, the procedure is the same as that for the first configuration in an installation. Teamcenter Environment Manager displays additional panels depending on the components you are including in the configuration.

Install Dispatcher using Deployment Center

The following procedures assume that you are installing Dispatcher components on an existing Teamcenter set up and that you are familiar with Deployment Center.

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

For more information about installing Teamcenter using Deployment Center, see *Teamcenter Installation Using Deployment Center*.

1. Log on to Deployment Center.
2. In the **Applications** task:

   a. Select **Teamcenter→Foundation→Extensions→Dispatcher**.

   b. Select **Teamcenter→Foundation→Enterprise Knowledge Foundation→Dispatcher Client for Rich Client**.

3. In the **Components** task:

   a. Select **Dispatcher Client (4 tier)** and specify the machine name and other parameters as appropriate.

   b. Select **Dispatcher Module** and specify the machine name and other parameters as appropriate.

      You can enable translators by selecting the required translators in the **Components** tab or manually enable the translators by editing the **translator.xml** file from the from the *Dispatcher_Root***Module\conf** directory.

      Select the required translators and specify the translator settings as appropriate. The translator settings are required for only specific translators.

   c. Select **Dispatcher Scheduler** and specify the machine name and other parameters as appropriate.

   d. Click **Save Component Settings**.

4. Generate deployment scripts.

Post-installation tasks

Create a dispatcher client access rule

After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

1. Log on to the Teamcenter rich client as a system administrator.

2. Open Access Manager.

3. In Access Manager, add the following rule to the rule tree under **Has Class (POM_application_object)→Working**:

   Condition =Has Class

   Value =DispatcherRequest

   ACL Name =DispatcherRequest

ACE Type of Accessor =User

ACE ID of Accessor =*DC-Proxy-User-ID*

ACE Privilege =Write

ACE Privilege Value =Grant

ACE Privilege =Delete

ACE Privilege Value =Grant

Note:

This rule applies only to the standard Access Manager rule tree. If custom rules are specified for the installation, your custom rules must be modified to provide the equivalent access of this rule. The *DC-Proxy-User* must be the same user that was specified during installation using Teamcenter Environment Manager.

Set up the common staging directory for translator input and output files

The RMI mode supports translations within a firewall and requires a common staging directory. In this mode, the Dispatcher Server clients can directly communicate with the scheduler and perform the translations without a web server and file server. The RMI mode does not require file transfers from the Dispatcher Server client locations and module locations.

The following are the prerequisites for setting up the staging directory:

- The clients and Dispatcher Server should point to the common staging directory and both should have read and write permissions to that directory.

  This avoids the need for file transfers from the client and module locations. The client writes the translator input files to the common staging directory, and the module writes the translator result files to the staging directory in the **results** directory. For example, if *d:\StagingDir* is the common staging directory for the module machine, then *\\ModuleMachine\StagingDir* on the client machine (can be a different machine) must be the same directory, that is, *StagingDir*.

- The client and Dispatcher Server should be installed by the same user to avoid directory access issues as the client and dispatcher modules use the same staging location.

- In the common staging directory configuration, the dispatcher module must have permissions to create directories and write access to files on the client staging location using the network.

  This means that when you access the staging location from the module machine, you should be able to access or list (in Linux systems) the directory contents of the staging location. The staging location can be a shared location on the network.

- The physical location of the machine on which the common staging directory is configured must be in closer proximity to the module machines than the dispatcher server client machines.

  This helps the translators to access files faster. The translators create temporary files and read the source files from the common staging directory.

Start Dispatcher services

During installation of the Dispatcher components, you specified the Dispatcher root directory. This directory is referred to as *DISP_ROOT*.

To fully process a Dispatcher request, all of the following services must be running in this order: scheduler, module, and dispatcher client.

During installation of the Dispatcher feature, you have the option of installing the Dispatcher services either as a service or as a console process.

Start scheduler

- Run the following command:

  Windows systems:

  DISP_ROOT`/Scheduler/bin/runscheduler.bat`

  Linux systems:

  DISP_ROOT`/Scheduler/bin/runscheduler.sh`

Start module

- Run the following command:

  Windows systems:

  DISP_ROOT`/Module/bin/runmodule.bat`

  Linux systems:

  DISP_ROOT`/Module/bin/runmodule.sh`

Start dispatcher client

- Ensure that the scheduler and module services are running before starting the dispatcher client.

  Run the following command:

Windows systems:

```
DISP_ROOT/DispatcherClient/bin/runDispatcherClient.bat
```

Linux systems:

```
DISP_ROOT/DispatcherClient/bin/runDispatcherClient.sh
```

Stop, pause Dispatcher services

If you have installed Dispatcher services as a service, you can stop them by using the **stop** command.

> Tip:
>
> The sequence for stopping services is the reverse of starting services and it is important to follow the sequence.

Stop dispatcher client

- Run the following command:

  Windows systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.bat -stop
  ```

  Linux systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.sh -stop
  ```

Stop module

- Run the following command:

  Windows systems:

  ```
  DISP_ROOT/Module/bin/runmodule.bat -stop
  ```

  Linux systems:

  ```
  DISP_ROOT/Module/bin/runmodule.sh -stop
  ```

Stop scheduler

- Run the following command:

Windows systems:

```
DISP_ROOT/Scheduler/bin/runscheduler.bat -stop
```

Linux systems:

```
DISP_ROOT/Scheduler/bin/runscheduler.sh -stop
```

Pause dispatcher client

- The pause command stops processing requests that are in the **INITIAL** state. This means the dispatcher client does not accept any new tasks. Tasks that are *In Progress* continue to be processed.

  To pause the dispatcher client, run the following command:

  Windows systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.bat -pause true
  ```

  Linux systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.sh -pause true
  ```

- To start the dispatcher client after pause, run the following command:

  Windows systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.bat -pause false
  ```

  Linux systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.sh -pause false
  ```

Set system and environment variables to run dispatcher client as a Windows service

On Windows systems, you can optionally configure the dispatcher client as a Windows service.

If you run the dispatcher client as a service, Siemens Digital Industries Software recommends running the scheduler and modules also as a service.

1. Set **TC_DATA** and **TC_ROOT** in the **System variables** section of the **Windows Environment Variables** dialog box.

   Examples:

```
TC_DATA=C:\Progra~1\Siemens\tcdata

TC_ROOT=C:\Progra~1\Siemens\Teamcenter8
```

Note:

When you install Teamcenter, TEM automatically sets the **FMS_HOME** system variable.

The Dispatcher Client service fails to start as a Windows service if you do not set the **TC_DATA** and **TC_ROOT** system variables.

2. Based on whether you use Teamcenter Key Manager, set the following environment variables:

   a. If you use Teamcenter Key Manager, set the **TC_KEY_MANAGER_PIPE** environment variable. You can find the value for this environment variable from the *TC_ROOT\install\tem_init.bat* file or the *TC_ROOT/install/tem_init.sh* file.

   b. If you do not use Teamcenter Key Manager, set the value of the **TC_USE_KEYMANAGER** environment variable to **false**.

3. Run the **runDispatcherClientWinService.bat** file from the **DispatcherClient\bin** directory.

4. From the Windows Services console, right-click **DispatcherClient***version* service and choose **Properties**.

5. In the **Log On** pane, choose the **This account** option to assign a log on account for the Dispatcher Client service.

   You must provide administrator privileges for the Dispatcher Client service.

6. Start the Dispatcher Scheduler.

7. Start the **DispatcherClient***version* service.

   Start the scheduler and modules before you start the dispatcher client to avoid connection delays and translation failures.

Delete and restart the Dispatcher client windows service prior to migrating to a new JRE or adding a custom JAR

You must delete the Dispatcher client windows service prior to migrating to a new JRE and restart it manually. The Java Runtime Environment (JRE) used by Teamcenter and TEM is set by TEM during Teamcenter installation. If you upgrade or install a new JRE, you must migrate Teamcenter to the new JRE using TEM.

Additionally, when you add a new custom JAR to the **DispatcherClient\lib** directory, you must delete and restart the Dispatcher client windows service.

To restart the Dispatcher client windows service, use the following command:

runDispatcherClientWinService.bat -r

View Dispatcher service runtime information

You can view the runtime status of Dispatcher services using the ping command.

> Note:
> The ping command is performance intensive; use it only when required.

View runtime status of scheduler

- Run the following command:

  Windows systems:

  ```
  DISP_ROOT/Scheduler/bin/runscheduler.bat -ping
  ```

  Linux systems:

  ```
  DISP_ROOT/Scheduler/bin/runscheduler.sh -ping
  ```

View runtime status of module

- Run the following command:

  Windows systems:

  ```
  DISP_ROOT/Module/bin/runmodule.bat -ping
  ```

  Linux systems:

  ```
  DISP_ROOT/Module/bin/runmodule.sh -ping
  ```

View runtime status of dispatcher client

- Run the following command:

  Windows systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.bat -ping
  ```

  Linux systems:

```
DISP_ROOT/DispatcherClient/bin/runDispatcherClient.sh -ping
```

View history of tasks performed by dispatcher client

- The history command parses the history logs and shows the detailed summary of tasks performed per day.

  Run the following command:

  Windows systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.bat -history
  ```

  Linux systems:

  ```
  DISP_ROOT/DispatcherClient/bin/runDispatcherClient.sh -history
  ```

Verify the Dispatcher installation

Verifying your Dispatcher installation

To verify that the Dispatcher installation was successful, check the following:

- The scheduler, module, and dispatcher client are installed, and you should be able to successfully start these services.
- If you installed dispatcher client for the rich client, you see the **Translation** menu in My Teamcenter.
- You are able to perform a test translation with the **tozipfile** translator.

> Caution:
>
> In the *TC_ROOT* directory, you see a directory called **Dispatcher**. You must not use the contents of this directory for Dispatcher customizations or configurations.
>
> All Dispatcher components are installed in a separate directory (*DISP_ROOT*). This directory is defined during Dispatcher installation. You must use this directory for Dispatcher customizations or configurations.

Verify Dispatcher components by performing a doc to zip file translation

Siemens Digital Industries Software recommends that you do a test **tozipfile** translation to make sure that Dispatcher components are working correctly.

Note:

Make sure that you have created a **dispatcher client access rule** and started the Dispatcher services.

1. In My Teamcenter, select an item revision.
2. Choose **File→New→Dataset** and in the **New Dataset** dialog box, select **MSWord** as the type for the new dataset.
3. In the **New Dataset** dialog box, click **Import** to import your Microsoft Word file.
4. In the **Import File** dialog box, select the Microsoft Word file you want to import into My Teamcenter and click **Import**.
5. Click **OK** to close the **New Dataset** dialog box.

   Your file is imported as the new dataset type.
6. Select the imported dataset and choose **Translation→Translate** to translate your Microsoft Word to the ZIP format.
7. In the **Translation Selection** dialog box, choose the **tozipfile** service.
8. Click **Finish** to start the translation process.
9. (Optional) Choose **Translation→ Administrator Console –All** to see the progress of the translation.

   After the translation is complete, the ZIP file appears in the item revision.

Test Dispatcher by translating files

1. In My Teamcenter, select an item revision.
2. In the item revision, select a CAD dataset and choose **Translation→Translate**.
3. In the **Translation Selection** dialog box, select the appropriate values for the **Provider** and **Service** lists.
4. Select the translation time, priority, and translation by repeating schedule options from the **Date and Time Properties** section.
5. Click **OK** to start the translation.

6. (Optional) Choose **Translation→Administrator Console –All** to see the progress of the translation.

7. After the translation is complete, the translated CAD file appears in the item revision.

If your site has the Teamcenter lifecycle visualization embedded viewer installed, you can view the translation result in the **Viewer** data pane in My Teamcenter or in the **Viewer** tab in Structure Manager.

Create translation requests in My Teamcenter or from the command line

You can create translation requests in My Teamcenter or from the command line.

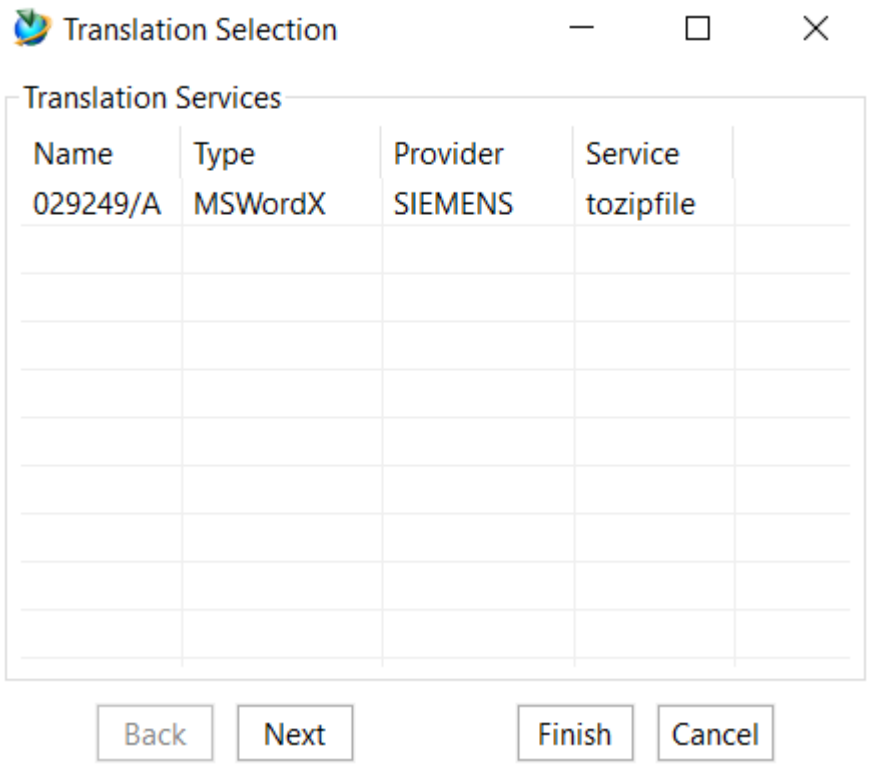
Create translation requests in My Teamcenter

1. In the navigation pane, select one or multiple datasets, item revisions, or structure context objects for translations.

2. Choose **Translation→Translate**.

   The **Translation Selection** dialog box shows the selected objects for translation.

3. In the **Translation Selection** dialog box, choose appropriate values from the **Provider** and **Service** lists.

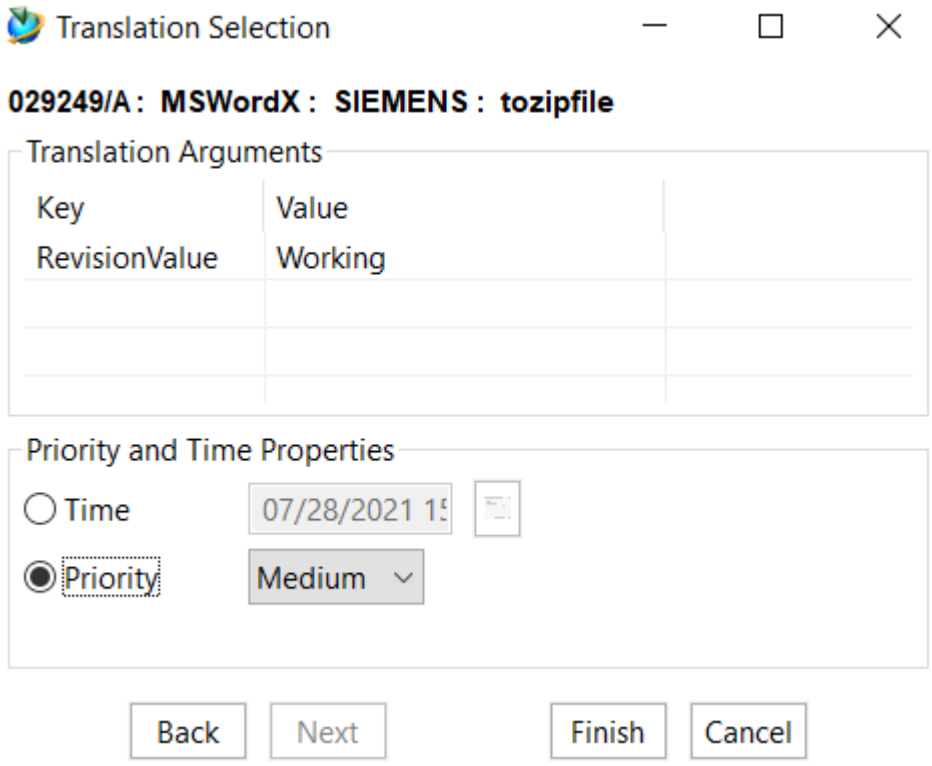


4. Click **Finish** to start the translation of all the objects.

   The default translator arguments are used for the translation.

5. If you want to specify translator arguments and other properties, click **Next**.

   Teamcenter shows the **Translation Selection** dialog box for the service.



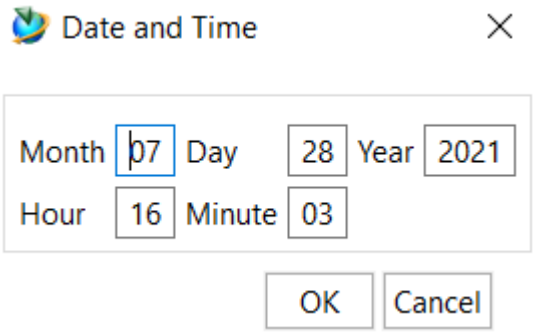
6. In the **Translation Arguments** section, you add, modify, or delete **Key** and **Value** arguments.

7. In the **Priority and Time Properties** section, you can set the following options:

   a. **Time**

      Choose the time for the translation to start.

      Click the **Admin Time and Date properties** button to display the **Date and Time** dialog box.



In the **Date and Time** dialog box, type the translation start time and click **OK**.

b. **Priority**

Choose the priority for the translation task.

c. **Repeating**

Choose this option if you want to repeat the translation.

> Note:
>
> The **Repeating** option does not appear by default. You must set the **ETS.Repeating_UI.<ProviderName>.<ServiceName>** preference to **TRUE** to display the repeating tasks functionality.

> Note:
>
> To avoid unpredictable behavior, the (time) interval in repeating tasks must be greater than the translation time.

8. Click **Finish** to start the translation.

   If there are other objects for translation, they are translated with the default values.

9. If you want to specify translator arguments and other properties for the remaining objects, click **Next**.

Create translation requests from the command line

You can create a translation request from the command line by using the **dispatcher_create_rqst** utility. This utility is located in the *TC_ROOT***/bin** directory.

You can get the usage details of this utility by typing the following on the command line:

```
dispatcher_create_rqst -h
```

Configure Dispatcher components

Enable Dispatcher to work with SSO

1. Install the dispatcher client using TEM. This ensures that Teamcenter creates a *Dispatcher-client-proxy-user* user.

2. Create the same *Dispatcher-client-proxy-user* user in LDAP.

3. Create an encrypted password file by running **encryptPass.bat/sh** *LDAP password for Dispatcher-client-proxy-user*.

4. Start dispatcher client.

Update Dispatcher components to work with UTF-8 configuration

When Teamcenter is set up to work in UTF-8, you can configure Dispatcher processes to support UTF-8. To do this, update the startup scripts for all Dispatcher components (Dispatcher Client, Scheduler, and Module) by adding the following JAVA VM option:

```
-Dfile.encoding=UTF-8
```

> Note:
>
> If your translator uses JAVA, you must update its startup script.

Switch from a 4-tier Dispatcher client to a 2-tier version

To switch from a 4-tier Dispatcher client to a 2-tier version:

1. Update the values of the *TC_ROOT* and *TC_DATA* entries in the *runDispatcherClient.bat* file or the *runDispatcherClient.sh* file.

2. Open the *DispatcherClient\conf\Service.properties* file and comment the following property:

   ```
   Tc.URL=http://localhost:7001/tc
   ```

Set preferences for repeating tasks

- Set the **ETS.Repeating_UI.<ProviderName>.<ServiceName>** preference to **TRUE** to display the repeating tasks functionality.

Caution:

Repeating task functionality is an intensive option. This functionality increases the load on the Dispatcher Server.

Enable log files for dispatcher requests

To attach log files to dispatcher requests, update the following value in the **Service.properties** file.

The **Service.properties** file is located in the *DISP_ROOT*/**DispatcherClient**/**Conf** directory.

- To attach log files to the dispatcher request when translation is successful, set

  Translator.*Provider name*.*Translator name*.**LogsForComplete** to **true**.

  Example:

  Translator.SIEMENS.tozipfile.LogsForComplete=true

  Note:

  By default, this attribute is set to false. If you do not want to attach log files to the dispatcher request (for successful translation), remove the entry from the **Service.properties** file.

- In case of translation failure, the log is automatically attached to the dispatcher request.

Using logs for error handling

When you install Dispatcher using TEM, you can specify a single root location for all log files. Logs are written in the following format for all the Dispatcher components.

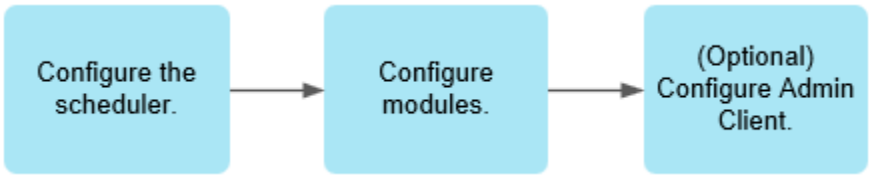
- *Log-volume-location*/*category*/**process**
- *Log-volume-location*/*category*/**task**

Log-volume-location is the central repository of log files and *category* refers to Dispatcher components. In addition, **process** is the directory for all process logs. Process logs are the main log files for Dispatcher components. Finally, **task** is the directory for all task logs. Task logs are log files specific to tasks such as submitting files for translation or generating translated files.

You can also view logs of a particular translation in the Dispatcher request administration console.

4. Configure the Dispatcher server

Configure Dispatcher Server



- *Configure the scheduler*

  To change the default values for the scheduler port, log volume location, and some other optional properties, edit the *DISPATCHER_ROOT***Scheduler\conf\transcheduler.properties** file.

- *Configure modules*

  For each module you have installed as per the deployment strategy, edit the following files:

  - To change the default values for the staging directory, scheduler URL, scheduler port, log files location, maximum number of tasks allowed in a module at a particular instant, and some other optional properties, edit the *DISPATCHER_ROOT***Module\conf\transmodule.properties** file.

  - To activate translators, edit the *DISPATCHER_ROOT***Module\conf\translator.xml** file.

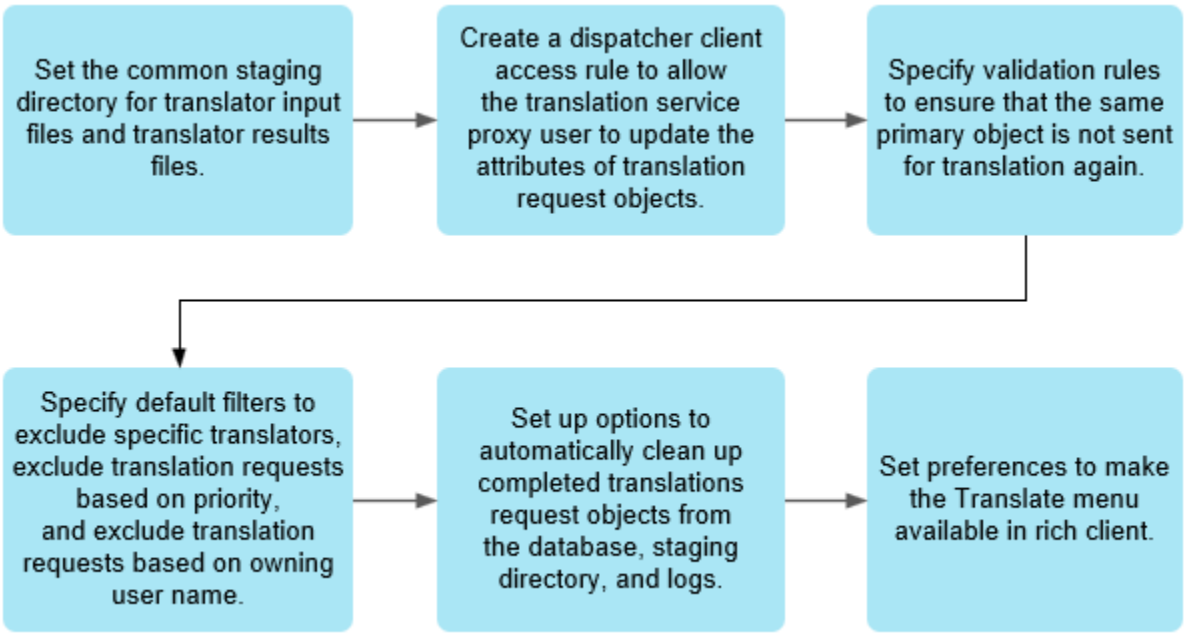
- *(Optional) Configure the Admin Client*

  To change the default values for the Dispatcher Server URL, staging directory, log files location, and some other optional properties, edit the *DISPATCHER_ROOT***AdminClient\conf\transclient.properties** file.

5. Configure the Dispatcher client for default translators

How to configure the Dispatcher client for default translators

The following is the typical workflow for configuring default translators or translators available with the installation kit.



- Set up the common staging directory for translator input and output files
- Create a dispatcher client access rule
- Create validation rules to disallow the same primary object from being translated twice
- Specify default filters to set translation rules
- Set up options to automatically clean up completed translations
- Enable translation options in the user interface by adding Dispatcher preferences

Create validation rules to disallow the same primary object from being translated twice

The dispatcher client does a duplicate check by default to make sure that the same primary object is not sent for translation. You can define translator-specific duplicate check by setting the **Service.CheckForDuplicateRequests** property in the property file of the translator. If this property is not set for the translator, the default **Service.CheckForDuplicateRequests** property value is used. Translator-specific duplicate check helps override the default **Service.CheckForDuplicateRequests** property.

> Example:
>
> Translator.*provider name*.*translator name*.Duplicate=false

Specify default filters to set translation rules

You can control translation services using dispatcher client filters. These provide a convenient way to control dispatcher request objects that are exposed to the dispatcher client for processing. If filters are specified for a dispatcher client, all dispatcher requests must pass this filter, or the dispatcher request is not processed by the dispatcher client.

Configuration of dispatcher client filter is specified in the **Service.properties** file, located in the *DISP_ROOT*/**DispatcherClient/conf** directory.

The Dispatcher client filters are implemented as a subclasses of the **RequestFilter** class. Dispatcher supplies three request filters:

- **TranslatorFilter**

  Filters dispatcher requests that do not contain the specified providers or translators.

- **PriorityFilter**

  Filters dispatcher requests that have a priority lower than the specified priority.

- **OwningUserFilter**

  Filters dispatcher requests that do not match the specified owning user name.

Set up options to automatically clean up completed translations

To clean up temporary files, use the **Service.RequestCleanup** options in the *DISPATCHER_ROOT***DispatcherClient\conf\Service.properties** file.

You can use the **Service.RequestCleanup** options to automatically cleanup completed translation request objects in the database, staging directory, and the logs. The system automatically deletes translation requests for both successful and unsuccessful translation states.

There are six options available: three each for setting the translation interval, translation threshold, and deleting logs for successful translations and three other options for the same settings for unsuccessful translations.

> Note:
>
> Deletion of the requests should be done by only one Dispatcher Client. Typically, the cleanup should be done with a Dispatcher Client which has less load. The rest of the Dispatcher Clients should only process requests.
>
> Use the **Service.RequestCleanup** property and disable the cleanup for all but *one* Dispatcher Client.

For examples, see the *DISPATCHER_ROOT***DispatcherClient\conf\Service.properties** file.

Configure the dispatcher client properties

The **Tc.URL** property is used to connect to Teamcenter to make SOA calls. By default, Dispatcher uses the 4-tier mode. For the 2-tier mode, you must comment out this property.

In addition, you can specify the Teamcenter group that the dispatcher client logs on as, the relogin interval, the owner of visualization datasets created by the dispatcher client, and other properties. For more information, see the *DISPATCHER_ROOT***DispatcherClient\conf\Service.properties** file.

Configure the dispatcher client for CAD dataset specific direct and indirect translators

Direct translators are connected to Teamcenter and can upload and download files without using any Dispatcher APIs. They handle the extraction of translation data and the loading of translated data back into Teamcenter. Examples of NX direct translators include **nxtocgmdirect**, **nxtopvdirect**, and **nxtransdirect**. These translators use Teamcenter user credentials specified through translator-specific configurations to perform translations and upload the translated files to Teamcenter.

Indirect translators are not connected to Teamcenter. Instead, they use **TaskPrep** class to extract and download the translation files and the **DatabaseOperation** class to upload the files back to Teamcenter. For more information, see **What is the extract-transform-load (ETL) model?** The Teamcenter user credentials specified in the *DISP_ROOT*/**DispatcherClient/Conf/Service.properties** file are used by the indirect translators for this process.

Note:

After uploading, Teamcenter applies a release status using the **ETS_released_status_type_names** preference. In the case of CAD dataset specific translators, the direct translators do not use the **Service.DataSetOwner**, **Service.StoreJTFilesInSourceVol**, and **Service.UpdateExistingVisualizationData** properties in the **Service.properties** file. Instead, they have their own logic to configure these properties and load the results back to Teamcenter. This is also true in the case of the **ETS_released_status_type_names** preference.

6. Enable default translators

Planning for enabling translators

Introduction to translators

Some system translators require a postprocessor translator and some require preprocessor translators. Typically, postprocessor translators are used to generate a 2D preview or thumbnail image of translated 3D data.

All presupplied translators require a postprocessor translator. These translators generate the 2D preview image of the 3D data.

Some authoring tools require different translators for 3D (part) and 2D (drawing) engineering data. Both formats must be translated to JPEG to create preview images. Typically, to support the full range of translations for a given tool, you must install, enable, and configure both the part and the drawing translators for that tool (in addition to the postprocessors).

Translators list

In your installation directory, refer to the *Dispatcher_Root***Module****Translators** directory for a complete list of translators. Each translator directory contains a *Readme.txt* file for the specific translator. This file provides a description of the translator, prerequisites, required licenses, and the installation instructions.

Enable translators

1. Run TEM or Deployment Center to install Dispatcher components as per your deployment strategy.

2. Run TEM or Deployment Center to install each translator on the appropriate module as per your deployment strategy.

   Alternatively, to manually configure the translators, locate the Dispatcher Server software distribution image for your platform and copy the **Translators** directory to the *Dispatcher_Root***Module** directory on each module machine.

   > Note:
   >
   > You must copy the **Translators** directory to the *Dispatcher_Root***Module** directory. The configuration assumes absolute and relative addresses based on this directory.

3. Install the required software as per specifications. For example, some CAD translators require the CAD software to be installed on the dispatcher machine.

You can follow the installation procedures provided by the software vendor to install the required software for CAD parts, CAD drawings, postprocessor and file translator.

> Note:
>
> Some CAD systems have a file path length limitation. The module must be installed in a root-level directory to avoid this problem.

4. Install the translator. Each translator has specific installation instructions. Refer to the related documentation for translator-specific installation instructions. Ensure that each translator is properly licensed.

5. Before integrating the translator with the Dispatcher Server, perform a manual test of the translation locally.

> Note:
>
> If the manual test is successful, it is easier to isolate issues specific to the file server, the scheduler, and the module.

6. Back up the **translator.xml** file, modify it to activate each installed translator, and then associate each translator with a translator module.

> Caution:
>
> Incorrect modifications to the **translator.xml** file can cause some or all translators to fail.

7. Some translators require **additional configuration**, such as setting batch file environment variables or tessellation configuration.

8. After implementing each translator, run the dispatcher module to include the translator so that you can progressively troubleshoot for any file translation errors.

9. Repeat the same steps for each translator you want to enable.

Editing the translator.xml file

Editing the translator.xml file

Each translator is associated with a translator module by specifying the required information in the **translator.xml** file in the *Dispatcher_Root***Module\conf** directory. Similar to any XML file, it consists of elements and attributes.

- Each translator is defined within a unique element

Each supported translator is defined in its own element (for example, **ToZipfile**), all nested within a **<Translators>** element. Nested within each translator element are elements that define the location of the translator and any required software, supported file name extensions, class path definitions, or any other required data.

- Modifying the **isactive** attribute to activate a translator

  To activate a translator, modify the **isactive** attribute to **true**. Retain it as **false** to keep the translator turned off.

  > Note:
  >
  > By default, the **isactive** value is set to **false** for all translators in the **translator.xml** file.

- Wrapper class attribute for defining a wrapper

  Most translators have a wrapper class attribute used to define a wrapper. Wrappers are used for managing translations including threading, process control (start and stop), status checking, error handling, and completion status.

  Example:

```
<ToZipfile provider="SIEMENS" service="tozipfile" maxlimit="1"
isactive="true" NoOfTries="1" OutputNeeded="true" MaximumProgress="100"
WaitTimeBetTrans="0" WaitTimeForReTries="0" ExclExitVal="1"
wrapperclass="com.teamcenter.tstk.translator.DefaultTranslator">
```

  > Note:
  >
  > You need not modify the **wrapperclass** attribute unless you write a custom wrapper to manage your translator.

- Modifying attributes to associate a translator

  To associate a translator with the translator module, modify the attributes that point to various executable files or the class path. In some cases, you must define additional elements to accommodate additional file extensions.

- Modifying input and output extensions

  The **InputExtensions** and **OutputExtension** elements define valid input and output file extensions for each translator.

> Note:
>
> If your site uses nonstandard file extensions (extensions that are not specified in the **translator.xml** file) for certain file types, you can add that file extension to the list of supported extensions.

Example:

```
<FileExtensions>
      <InputExtensions nitem="1">
        <InputExtension extension=".pkg"/>
      </InputExtensions>
      <OutputExtensions nitem="2">
        <OutputExtension extension=".jt"/>
        <OutputExtension extension=".cgm"/>
      </OutputExtensions>
    </FileExtensions>
```

As shown in the example above, you must change the **nitem** attribute based on the number of input or output file extensions for each translator.

- Editing the **maxlimit**, **NoOfTries**, and **OutputNeeded** properties.

  - **maxlimit** defines the maximum number of translation that can run simultaneously. The default value is **1**.

  - **NoOfTries** defines the number of tries to make in case the translation fails. The default value is **1**, that is, no retry.

  - **OutputNeeded** defines whether a result file is expected from the translation. The default value is **true**.

Define entities

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Locate the **<!ENTITY>** elements located at the start of the file.

3. For each translator you want to activate, set the **isactive** attribute to **true**.

   Example:

```
<!-- Configuration of the Pro Engineer to JT translator -->
<ProEToJt provider="Siemens" service="proetojt" isactive="true"
          wrapperclass="&EAIWRAPPER;">
```

4. For each installed translator, verify the **MODULEBASE** property.

   Example of a translator startup file called from the *Dispatcher_Root***Module\Translators** directory:

   ```
   <TransExecutable dir="&MODULEBASE;/Translators/catiatojt/v5/UGS"
   name="catiav5tojt.sh" />
   ```

   Example of a deamon executable called from the directory where the translator is installed:

   ```
   <Daemon exec="/proe<ver>/sun4_solaris/nms/nmsd" args="-noservice"/>
   ```

Modify input extensions

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Locate the **<InputExtensions>** tag for the translator you want to modify. Increment the value of the **nitem** parameter to the total number of file extensions you want to support.

3. For each additional file extension, add an **<InputExtension extension=" ">** tag. Specify the file extension (including the dot) in the **extension** attribute.

The following example shows the **IdeasToJt** translator definition modified with the **<InputExtension extension="">** tag to accept files with the **SLDASM** extension:

```
<FileExtensions>
      <InputExtensions nitem="1">
        <InputExtension extension=".pkg"/>
      </InputExtensions>
      <OutputExtensions nitem="2">
        <OutputExtension extension=".jt"/>
        <OutputExtension extension=".cgm"/>
      </OutputExtensions>
    </FileExtensions>
```

Creating an exclusion list for error messages

Creating an exclusion list for error messages

Translators display error messages or warning messages while performing translations. The dispatcher module cannot distinguish between serious errors that result in translation failures and simple warnings. For example, the following warning can prevent the dispatcher module from completing the translation:

```
This is not an error string
```

The Dispatcher Server provides an exclusion list in the **translator.xml** file to ignore simple warnings and continue the translation. The **translator.xml** file contains predefined **InputStream** and **ErrorStream**

error strings with keywords such as *cannot*, *error*, *exception*, and *failed*. The dispatcher module checks both the streams for the error displayed by the translator, verifies whether the same error is defined in the exclusion string, and if they are the same, the module continues with the translation.

You can add specific error strings to the exclusion list to prevent the dispatcher module from stopping translations. For example, you can add **Not a Failure** within the **TransErrorExclStrings** tags of the **ToZip** translator in the translator.xml file to prevent this translator from failing.

Error strings

The error tags can be either translator specific or global. You can use:

- *Translator-specific* error tags for specific translators defined in the **translator.xml** file.
- *Global* error tags for all active translators in the dispatcher module.

> Note:
>
> When you use translator-specific error tags, the dispatcher module ignores global error strings.

Sample translator-specific error tags:

```
<UgToPvDirect provider="Siemens" service="ugtopvdirect"
       maxlimit="1" debug="&DEBUG">
<TransExecutable name="ugtopvdirect.sh" dir="&MODULEBASE;/Translators/
ugtopvdirect"/>
       <Options>
       <!-- Option name="" string="" value="-single_part"
       description=""/ -->
       <Option name="ConfigMapFile" string="-configmap"
       value="&MODULEBASE;/Translators/ugtopvdirect/
       ConfigMap.properties"
       description="Property file which has the Map between Client
       passed in Criteria and the location of config file to be used"/>
  <Option name="inputpath" string=""
      description="Full path to the input file."/>
  </Options>
     <TransErrorStrings>
        <TransInputStream string="Cannot"/>
        <TransInputStream string="Error"/>
        <TransInputStream string="exception"/>
        <TransInputStream string="ERROR"/>
        <TransErrorStream string="Errored"/>
        <TransErrorStream string="failed"/>
     </TransErrorStrings>
  <!-- Postprocess provider="Siemens" service="copyugrelstatus"/ -->
  </UgToPvDirect>
```

Sample global error tags:

```
<ErrorStrings>
    <InputStream string="Error"/>
    <InputStream string="error"/>
    <InputStream string="Null"/>
    <InputStream string="Exception"/>
    <InputStream string="ERROR"/>
    <ErrorStream string="Error"/>
    <ErrorStream string="error"/>
    <ErrorStream string="Exception"/>
    <ErrorStream string="termination"/>
    <ErrorStream string="Bad"/>
    <ErrorStream string="Null"/>
    <ErrorStream string="ERROR"/>
  </ErrorStrings>
```

Exclusion tags

Similar to error tags, the exclusion tags can be either translator specific or global. You can use:

- *Translator-specific* exclusion tags for specific translators defined in the **translator.xml** file.
- *Global* exclusion list for active translators in the dispatcher module.

> Note:
>
> When you use translator-specific exclusion tags, the dispatcher module ignores the global exclusion list.

Sample translator-specific exclusion tags:

```
<!-- Configuration of the To Zip translator -->
  <ToZipfile provider="Siemens" service="tozipfile" maxlimit="1"
             debug="&DEBUG">
    <SysExecutable name="java"/>
    <Options>
      <Option name="classpath" string="-cp" system="java.class.path"
             description="Classpath to the java virtual machine." />
      <Option name="classname" string=""
             value="com.sdrc.intellivis.translator.ZipUtility"
             description="Zip Utility class."/>
      <Option name="zip" string="-zip" value=""/>
      <Option name="inputpath" string=""
             description="Full path to the input file or directory."/>
      <Option name="outputpath" string=""
             description="Full path to the output zip file."/>
    </Options>
```

```
    <TransErrorStrings>
      <TransInputStream string="Error"/>
      <TransErrorStream string="Null"/>
    </TransErrorStrings>
    <TransErrorExclStrings>
      <TransInputStreamExcl string="Not a Failure"/>
      <TransErrorStreamExcl string="Translator did not fail."/>
    </TransErrorExclStrings>
    <FileExtensions>
      <OutputExtensions nitem="1">
        <OutputExtension extension=".zip"/>
      </OutputExtensions>
    </FileExtensions>
  </ToZipfile>
```

Sample global exclusion list tags:

```
<!-- Translator input and error stream keywords to
       exclude as error conditions. -->
  <!-- These are common to all translators. Translator specific
       keywords must be-->
  <!-- provided in the configuration of the specific translator.-->
  <ErrorExclStrings>
    <InputStreamExcl string="Error Reporting mechanism"/>
    <ErrorStreamExcl string="This is not an error"/>
  </ErrorExclStrings>
```

Modifying tessellation configuration files

Tessellation is a process that translates a CAD file to a visualization file.

The table that follows describes the translators for which you can modify tessellation output options.

> Note:
>
> If you do not specify any options, the translator service generates a default output.

Translator	Modify which file?
CATIA 4 Part Translator (Theorem) **Catiav4ToJt**	**.bat** or **.sh** script of the translator to point to the tessellation configuration file of the translator
CATIA 4 Part Translator (UGS) **Catiav4ToJt**	**.sh** script of the translator to point to the tessellation configuration file of the translator[1]

1 This translator is available only in Linux.

Translator	Modify which file?
CATIA 5 Part Translator **Catiav5ToJt**	*Dispatcher_Root***Module\conf\translator.xml**
NX Part Translator **NxToPv**	**.bat** or **.sh** script of the translator

Note:

You must edit either the **translator.xml** file or the startup script (**.bat** or **.sh**) for each translator to ensure that the Dispatcher Server uses the latest tessellation options.

Enable the default translators

Enable asynchronous services

Configuring asynchronous services

The **AsyncService** translator independently processes asynchronous requests constructed by the Teamcenter server using Teamcenter SOA native C++ framework in the background mode.

Background operations are accomplished by using Dispatcher to hold and schedule the asynchronous request and to initiate the operation as a standard services oriented architecture (SOA) request in a separate Teamcenter four-tier session. Operations are executed using the Teamcenter middle tier for session management and load balancing.

The operation can be executed either on the local site or on a remote site. When running the operation on a remote site, the asynchronous facility uses the same configuration and authentication information in the site object as used by Teamcenter Multi-Site.

If the site uses Teamcenter Security Services single sign-on (SSO), the asynchronous facility uses SSO tokens to authenticate to the remote site. After the request is completed, the Teamcenter event or notification, or subscription service is used to inform the user that the background task is complete.

The asynchronous session is initiated in a new session as the same user, group, or role and locale as the original session.

- Authentication without Security Services

  The original Teamcenter server session generates a unique authentication token for the user, encrypts it, and stores it with the background request. When the request is executed, this token is decrypted and used in place of a password to authenticate the user's session.

- Remote invocation

The asynchronous facility can execute the operation on a remote site if the installation is configured for Multi-Site.

Configuring the AsyncService

1. Set up a four-tier environment.

2. Set **SOA URL** for your site.

   Use the Organization application to create a site and add the middle tier system's URL to the **SOA URL** field in the site object of your local site and each remote site to which you wish to send asynchronous requests. This should have the same base value as used to set up and configure the middle tier and to run Active Workspace and the rich client.

   Example:

   http://localhost:7001/tc

   If you use SSL to access Teamcenter, that is, if the address starts with **"https:"**, ensure that a valid certificate is stored in the business logic server's trust certificate store file.

3. **Configure the dispatcher**.

   Install Dispatcher and set up the dispatcher client, scheduler, and module.

4. (Optional) **Configure Security Services**.

   The asynchronous facility supports Security Services for authentication of the asynchronous session. When calling requests on a different site, both the calling and destination site must use the same Teamcenter Security Services directory. In addition, you must configure Teamcenter and Security Services to define a shared mediator key.

5. **Enable the AsyncService**.

   The **AsyncService** translator independently processes asynchronous requests constructed by the server using the Teamcenter service-oriented architecture (SOA) native C++ framework in background mode.

6. **Configure the async_invoker service**.

   You must set up the operating system user name on which you run the Dispatcher module that runs the **AsyncService** service to execute Teamcenter utilities in autologon mode. That is, make sure there is a Teamcenter user who has the same OS user name setting as the operating system user running the module service.

Configuring Dispatcher

- Install Dispatcher and set up the dispatcher client, scheduler, and module.
- Set the **maxlimit** parameter to run multiple requests.

  By default, the dispatcher module runs only one request of a particular type at a time. This limits your throughput as users may submit many requests. However, ensure that you do not allow more requests than what your server manager pool can handle at a time. To configure this value, add the **maxlimit** parameter to the **AsyncService** element in the *Dispatcher_Root***Module\conf\translator.xml** file.

  Example:

  ```
  <AsyncService provider="SIEMENS" service="asyncservice"
  maxlimit="2" isactive="true">
  ```

- Clean up temporary files.

  To clean up temporary files, use the **Service.RequestCleanup** options in the **DispatcherClient/conf/Service.properties** file.

  For more information, see the **DispatcherClient/conf/Service.properties** file.

Configuring Security Services

The asynchronous facility supports Security Services for authentication of the asynchronous session. When calling requests on a different site, both the calling and destination site must use the same Teamcenter Security Services directory. In addition, you must configure Teamcenter and Security Services to define a shared mediator key.

A type of Teamcenter Security Services token is used only in conjunction with mediating applications. Mediating applications (such as Teamcenter when invoking the **AsyncService** service) can assume the role of a Security Services session agent and submit special Security Services log on requests. All log on requests to Security Services return a Teamcenter Security Services application token built for the target application, but the token that Teamcenter Security Services returns to the mediating application has a special structure: It contains an inner token, which is intended for the target application. This token is returned to the mediating application wrapped in an outer token that is separately encrypted. The mediating application decrypts the outer token and extracts and forwards the inner token to the target Teamcenter Security Services application, which subsequently validates that token back with Security Services.

> Note:
>
> Configuring asynchronous services with Security Services depends on a valid security services session that has not been affected by a timeout or expiration.

To configure Teamcenter Security Services application tokens:

- Set a mediator password in Security Services (using Web Application Manager while building Security Services).

- Set the same mediator password for Teamcenter (using install_encryption_keys).

  Run the **install_encryptionkeys** utility as follows, and enter the mediator password when the utility prompts for the password:

  install_encryptionkeys -u=*user_name* **-p=***password* **-g=***dba* **-f=***install_mediator_key*

If the default one week lifetime is not adequate, set the **ASYNC_credentials_lifetime** site preference.

When the server calls an asynchronous request, it obtains an special double-encrypted token from the Security Services Identity Service and stores it in the **DispatcherRequest** along with the other information for the request. When the Dispatcher schedules and invokes the request, the **AsyncService** service uses the mediator key to decrypt the token and uses it to log on to the new Teamcenter session as the original user.

Configure HTTP enabled Multi-Site for single sign-on

If you use Security Services single sign-on (SSO) functionality, configure the **TC_SSO_app_id_of_site_***site-name* preference at the site. Here, *site-name* represents the name of the site, and this preference does not exist by default. The preference value must match the **Application ID** value for the site as defined in the Application Registry table.

> Note:
>
> All sites that are using SSO must be in the same SSO domain.

Enable the AsyncService translator

The **AsyncService** translator independently processes asynchronous requests constructed by the server using Teamcenter service-oriented architecture (SOA) native C++ framework in background mode.

Skip these steps if you use TEM or Deployment Center to enable the translators.

1. Open the **translator.xml** file in the *Dispatcher_Root***\Module\conf** directory.

```
<AsyncService provider="SIEMENS" service="asyncservice"
isactive="false">
    <TransExecutable name="asyncservice.bat" dir="&MODULEBASE/
Translators/
     asyncservice"/>
    <Options>
      <Option name="inputpath" string=""
              description="Full path to the input file or
directory ."/>
```

```
    <Option name="outputdir" string=""
            description="Full path to the output file."/>
    <Option name="OutputFileName" string="" value="output.txt"
            description="Name of the output file."/>
  </Options>
  <TransErrorStrings>
    <TransInputStream string="AsyncInvoker Report"/>
    <TransErrorStream string="AsyncInvoker Report"/>
  </TransErrorStrings>
</AsyncService>
```

2. Set the **isactive** attribute to **true** to activate this translator.

3. Edit the **CHANGE_ME** tags in the **asyncservice.bat** (Windows) or **asyncservice.sh** (Linux) file in the *Dispatcher_Root***Module\Translators\asyncservice** directory.

   For more information, see the respective files.

4. Create the input required for this translator.

   Example 1:

   Create an item revision, right-click the item revision, and send it to Structure Manager. Select **File→Duplicate** and select the **Run in Background** option.

   Example 2:

   Create a workflow task in Workflow Designer. When you create this workflow task, be sure to select the **Process in Background** option in the **Attributes** dialog.

   The system creates an **input.txt** file. You can use this file as the input for this translator.

5. From the command prompt, run the translator in standalone mode to verify whether it is working properly.

   For information about running the translator in standalone mode, from a command prompt, change to the *Dispatcher_Root***Module\Translators\asyncservice** directory and call **asyncservice.bat -help** (Windows) or **asyncservice.sh -help** (Linux).

   To run this translator in standalone mode, you must specify the full path name of the file that holds the async request arguments by editing the **asyncservice.bat** file.

   Example:

   ```
   asyncservice.bat C:\Dispatcher\input.txt C:\Dispatcher\result\output.txt
   ARG1 is a full path name of the file which holds the async request
   arguments.
   ```

```
ARG2 is the location of the result files (log, output, etc).
ARG3 is the output file name that the async invoker program
generates.
```

Where **C:\Dispatcher\input.txt** is the full path name of the file used as the input for this translator and is generated by running the workflow task (step 4) and **C:\Dispatcher\result\output.txt** is the output file that the **AsyncService** translator generates.

Configure async_invoker service

Set up automatic log on for async_invoker service

You must set up the operating system user name on which you run the Dispatcher module that runs the **AsyncService** service to execute Teamcenter utilities in autologon mode. That is, make sure there is a Teamcenter user who has the same OS user name setting as the operating system user running the module service. This Teamcenter user needs no special privileges and can be a new unique Teamcenter user or an existing administrative user.

- Use the Organization application to create or find this user, and specify a valid **OS Name** field for the Teamcenter user.
- Use this operating system name to log on to Windows or Linux, or to set up the log on information for the Windows service for the Dispatcher module that runs the **AsyncService** service.
- Test the autologon mode by running a Teamcenter utility called **list_users** without providing a user name or password.

> Note:
>
> This ensures that the Dispatcher module does not prompt for the Teamcenter user name and password. If autologon is not working, the **AsyncService** service hangs waiting for input from the non-interactive standard input stream and the Dispatcher module cannot complete the request.

Configure the user name and password file log on

Configure the user name and password file log on.

In the *Dispatcher_Root***Dispatcher\Module\Translators\asyncservice\asyncservice.bat** file (**asyncservice.sh** file for Linux) add the **-u** and **-pf** parameters to the end of the line that invokes the **asyncinvoker** program.

Windows example:

```
"async_invoker.exe" %1 %2 %3 %4 %5 -u=tcadmin
-pf=C:\teamcenter_creds\pwf.txt
```

> Note:
>
> Ensure that password files are managed well and private. Only users who run Teamcenter servers, in this case, the Dispatcher module, should have access to password files.

In a Teamcenter Security Services environment, remove or comment the following lines to allow Teamcenter Security Services authentication to take place:

```
rem  disable SSO for asyncservice so that auto-login can complete.
set TC_SSO_APP_ID=
set TC_SSO_SERVICE=
set TC_SSO_LOGIN_URL=
```

Configure connection retries

If the **async_invoker** service cannot connect to the four-tier environment, it reattempts a configurable number of times (60), at a configurable interval (60 seconds), before failing. To change the retry count or interval, set the following preferences:

```
preferences_manager -u=user–id -p=password -g=dba -mode=import
-preference=ASYNC_connection_retries -scope=SITE -values=1
-action=OVERRIDE

preferences_manager -u=user–id -p=password -g=dba -mode=import
-preference=ASYNC_connection_retry_interval -scope=SITE -values=10
-action=OVERRIDE
```

Enable PLM XML based translators

Enable the CreateAssemblyPLMXML translator

The **CreateAssemblyPLMXML** dispatcher task creates a **PLMXML** file of the configured assembly and loads it in a dataset related to the root item revision. The dispatcher task invokes the controlled replication mechanism to replicate the configured assembly.

> Note:
>
> Teamcenter and the Multi-Site Collaboration feature must be installed to use this translator.

Enable the translator in a non-SSO environment

Skip these steps if you use TEM or Deployment Center to enable the translators.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

3. Edit the **CHANGE_ME** tags in the **createassemblyplmxml.bat** or **createassemblyplmxml.sh** file in the *Dispatcher_Root***Module\Translators\createassemblyplmxml** directory.

   For more information, see the respective files.

4. From the command prompt, run the translator in standalone mode to verify whether it is working properly.

   For information about running the translator in standalone mode, from a command prompt, change to the *Dispatcher_Root***Module\Translators\createassemblyplmxml** directory and call **createassemblyplmxml.bat -help**.

Enable the translator in an SSO environment

1. Stop Dispatcher services.

2. Choose a DBA user, for example, **user** (*Dispatcher-client-proxy-user* is the default user).

3. Create a password *abc.txt* file containing the password for the *Dispatcher-client-proxy-user* user.

   The default password file for the *Dispatcher-client-proxy-user* user is *TC_BIN***PasswordFile.txt**.

4. Edit the following files:

   - **createassemblyplmxml\createassemblyplmxml.sh** (Linux) or **createassemblyplmxml\createassemblyplmxml.bat** (Windows)
   - **replicateplmxml\replicateplmxml.sh** (Linux) or **replicateplmxml\replicateplmxml.bat** (Windows)
   - **plmxmlbasedsync\plmxmlbasedsync.sh** (Linux) or **plmxmlbasedsync\plmxmlbasedsync.bat** (Windows)
   - **importobjects\importobjects.sh** (Linux) or **importobjects\importobjects.bat** (Windows)

5. Change as follows.

   - Linux example: *TC_ROOT*/**bin/data_share -u=***Dispatcher-client-proxy-user* **-pf=***TC_ROOT/PasswordFile.txt* **-f=***remote_import*

     Change to:

     TC_ROOT/**bin/data_share -u=***user* **-pf=***/usr/securepwds/abc.txt* **-f=***remote_import*

   - Windows example: *TC_ROOT***bin\data_share.exe -u=***Dispatcher-client-proxy-user** **-pf=***TC_ROOT\PasswordFile.txt* **-f=***remote_import %**

Change to:

TC_ROOT\bin\data_share.exe -u=*user* -pf= *C:\dispatchpwds\abc.txt* -f=*remote_import %**

6. Start Dispatcher.

Enable PLMXMLBasedSync, plmxmltojt, plmxml_import_export, and ReplicatePLMXML translators

- **PLMXMLBasedSync** service

  The **PLMXMLBasedSync** dispatcher task parses the **PLMXML** file of a configured assembly, identifies item revisions that need sync or import and performs the import as per the controlled replication mechanism.

  Prerequisites:

  Teamcenter and the Classic-Multisite feature must be installed to use this translator.

- **plmxmltojt** translator

  Translates PLMXML files (**.plmxml**) to JT (**.jt**).

  Prerequisites:

  Teamcenter Lifecycle Visualization must be installed.

- **plmxml_import_export** translator

  Imports and exports objects into Teamcenter asynchronously through the rich client. Currently, it supports only exporting objects. The translator uses the **plmxml_export** utility in the background to transfer objects.

  During export, the system replaces the reserved key characters such as space, comma, semicolon, tab, and equal in the target file name with the underscore (_) character.

- **ReplicatePLMXML** service

  Executes data_share -f=remote_import for a given items list.

  Prerequisites:

  Teamcenter Engineering and Classic-Multisite feature must be installed.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

   For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

   > Note:
   >
   > For more information, see the **Readme** file in the **Translators***translator_name* directory.

   - Windows: *translator_name*.**bat -help**
   - Linux: *translator_name*.**sh -help**

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file.

   NxTransDirect translator example:

   ```
   "%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=Dispatcher-client-proxy-user
   -p=password
   -changeOwnerToCad=false "-pim=yes" %*
   ```

   If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update the attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

Enable BatchPrint and PublishBatch translators

- **BatchPrint** service

  The **BatchPrint** Dispatcher client handles calls from the rich client to process the batch printing of files. This translator processes batch print requests submitted by the **BatchPrint** Dispatcher client. The **batchprint.bat** (for Windows) or **batchprint.sh** (for Linux, Linux, and Mac OS) file invokes the Vis Print application to print the selected files.

  Prerequisites:

  - **Vis Home Path** location, that is, the path to the visualization directory while installing **BatchPrint** using TEM or Deployment Center.
  - **Vis License** location while installing **BatchPrint** using TEM or Deployment Center.

- **PublishBatch** translator

  Prerequisites:

  The Dispatcher infrastructure for scheduling create and update features.

  - The translator is supported on Windows only.
  - VisAutomationApp, Microsoft Visio, FMS file client cache (FCC), and the Teamcenter SOA client installed on the translator machine.

    > Note:
    >
    > The VisAutomationApp is installed automatically when you install Lifecycle Visualization.

    For more information about installing Lifecycle Visualization and MS Visio, see the *Teamcenter lifecycle visualization Install Guide* and the *Microsoft Visio Online Help*, respectively.

  - The **startPublishBatch.bat** file is installed in a directory that is specified during installation of the **PublishBatch** translator. Each setting in this file must be modified as specified in the file.
  - The **transmodule.properties** file in the **module/conf** directory has a setting called **MaximumTasks** that defaults to **3**. This defines the maximum number of instances of publish batch tasks that can run simultaneously. Do not increase this value above **3**.

- The clients and Dispatcher Server should point to the common staging directory with read/write permissions to the directory.

- (Optional) Open the **startPublishBatch.bat** file and add the **–log** option to the *TRANS_PATH***tcpublish_batch** attribute. Teamcenter then adds detailed progress and error entries to the task's log file.

  The **startPublishBatch.bat** file is installed in a directory that is specified during installation of the **PublishBatch** translator. Follow the instructions in the file to add the **–log** option. Teamcenter stores the log files in the directory you specify during installation.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator.

The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name* directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

Enable CAD part translators

- *Catiav4ToJt translator*

  Translates Catia V4 (**.model**) files to JT (**.jt**) files.

- *Catiav4ToJt translator (Linux)*

  Translates Catia V4 (**.model**) files to JT (**.jt**) files.

  The CATIA 4 part translator (Siemens) is supported only on Linux systems.

- *Catiav5ToJt translator*

  Translates Catia V5 (**.CATPart**) files to JT (**.jt**) files.

After you enable the **Catiav5ToJt** service, this service is not available from the default **Translate** menu in My Teamcenter. There is a separate **CATIA V5** menu that you can use to perform a conversion to JT files.

- *JtToCatia translator*

  Translates JT (**.jt**) files to Catia (**.CATPart** and **.CATProduct**) files.

- *ProEToJt translator*

  Translates Pro/ENGINEER drawing (**.drw**) files to DXF (**.dxf**) files. It supports all Pro/ENGINEER drawing files.

  Prerequisites for this translator:

  - Norton AntiVirus may cause Pro/ENGINEER translators to hang. Disable it on the translation module that includes the Pro/ENGINEER translator.
  - Do not install the **Vis Mockup** translator on any dedicated translation module machine used to run the **ProEToJt** translator as these translators cause conflicts.
  - If the Pro/ENGINEER translator is installed in a directory with spaces in the directory name (for example, **Program Files**), ensure that the value of the **PROE_CMD** variable in the **proetojt***version*.**bat** file is set in quotation marks. For example:

    ```
    PROE_CMD="D:\Program Files\proe<ver>\bin
      \proe<ver>.bat”
    ```

- *SolidWorksToJt translator*

  Translates SolidWorks part and assembly files to the Direct Model (**.jt**) visualization format.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator.

The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

Note:

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

3. Open the *translator***.bat** (Windows) or *translator***.sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

Enable CAD drawing translators

- *NxToCgmDirect translator*

  Translates UG (**.prt**) files to CGM (**.cgm**) files. It connects to Teamcenter to perform the translations.

  Prerequisites:

  - This is not supported on four-tier or SSO environments.
  - Ensure that a matching user name is available in Teamcenter for the user name of the user running the translator module. This user must have all read and write and checkin and checkout privileges to perform translations.

    Note:

    To overcome this limitation, modify the *Dispatcher_Root***Module\Translators\nxtocgmdirect\nxtocgmdirect.bat** (Windows) or *Dispatcher_Root***Module\Translators\nxtocgmdirect\nxtocgmdirect.sh** (Linux) file to add the **-u**, **-p**, or **-pf** arguments.

- *NxToPv translator*

  Translates UG (**.prt**) files to JT (**.jt**) files.

- *NxToPvDirect translator*

  Translates UG (**.prt**) files to JT (**.jt**) files.

  Prerequisites:

  - This is not supported on four-tier or SSO environments.
  - Ensure that a matching user name is available in Teamcenter for the user name of the user running the translator module. This user must have all read/write and checkin/checkout privileges to perform translations.

> Note:
>
> To overcome this limitation, modify the *Dispatcher_Root***Module\Translators\nxtopvdirect\nxtopvdirect.bat** (Windows) or *Dispatcher_Root***Module\Translators\nxtopvdirect\nxtopvdirect.sh** (Linux) file to add the **-u**, **-p**, or **-pf** arguments.

- *ProEToDxf translator*

  Translates the Pro/ENGINEER drawing files to the Autodesk Drawing eXchange Format (**DXF**) format.

  The batch file that starts the Pro/ENGINEER translator may have different names depending on the installation. The Dispatcher Server looks for **InvokeProE.bat** file in the **bin** directory to invoke the translator. Therefore, if the batch file has a different name, you must save the file as **InvokeProE.bat** in the **bin** directory for the Dispatcher Server to invoke the Pro/ENGINEER translator.

  Create a **PRO_COMM_MSG_EXE** environment variable that points to *loadpoint*/*OS name***/obj/pro_comm_msg.exe**.

  For example, set PRO_COMM_MSG_EXE to:

  ```
  C:\ptc\proe2000i2\i486_nt\obj\pro_comm_msg.exe
  ```

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator***.bat** (Windows) or *translator***.sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

Enable the ContMgmtPublish translator

The **ContMgmtPublish** translator is used to render Content Management content to an output file format defined by a publishing tool, such as PDF or HTML. The translator then stores the output file in Teamcenter.

> Note:
>
> You can install this translator using TEM or Deployment Center only. This translator is not designed to run in a standalone mode or run by a user directly.

1. Install the required XSL formatter for this translator, such as Antenna House.

2. (Optional) If you use a DITA open toolkit to publish content, install Java JDK version 1.7 and modify the *Dispatcher_Root***Module\Translators\contmgmtpublish\-config\contmgmtpublish_config.properties** file to change the **JDK_HOME** variable to the JDK installation directory.

   > Note:
   >
   > - Content Management supports version 2.4.6 of the DITA Open Toolkit (OT) publishing tools.
   > - To use an older version of DITA OT, copy the older version of DITA OT to: the *Dispatcher_Root***Module\Translators\contmgmtpublish\lib** folder.
   > - Set the path for `DITA_ANT_HOME` in the *Dispatcher_Root***Module\Translators\contmgmtpublish\config\-contmgmtpublish_config.properties** file. The path must be set to the Ant directory in the DITA OT.

3. To verify the installation, after you set up the **ContMgmtPublish** translator, publish a topic revision using Teamcenter Content Management rich client (**Tools→Publish Content**).

Enable the translator

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

> Note:
>
> By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module****Translators***translator_name* directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

Enable FMSTransfer, ImportObjects, JtToBBoxAndTso, and mmvindexgen translators

- **FMSTransfer** service

  Used to forward a file from a temporary store and forward volume to its destination volume.

  Prerequisites:

  - **TC_ENABLE_STOREANDFORWARD** site preference must be set to **true**.
  - Uploaded files are transferred to their destination according to the set rules and the configured schedule.
  - The user must have appropriate permissions to upload files.

- **ImportObjects** service

  Executes data_share -f=remote_import for a given items list.

  Prerequisites:

  Teamcenter and the Classic-Multisite feature must be installed to use this translator.

- **JtToBBoxAndTso** service

  Generates bounding box and TSO files from JT files in Teamcenter.

  Prerequisites:

  - Teamcenter and the **jttobboxandtso** utility must be installed to use this service.
  - This translator requires the Teamcenter password.

To encrypt a password file, you set a temporary environment variable to the password you want to encrypt, and then generate an encrypted password file using the **-encryptpwf** argument for the install utility.

- **mmvindexgen** translator

  An MMV dataset consists of a spatial hierarchy of a model, which is harvested from JT data using the Teamcenter Dispatcher **mmvindexgen** translator. You can use the My Teamcenter **Translation** menu to automate the generation of the spatial index on a recurring basis, in order to capture design changes over the course of the product lifecycle.

  Prerequisites:

  - When MMV index production is performed on an assembly under which a **Spatial Hierarchy** dataset already exists for one or more item revisions, the existing **Spatial Hierarchy** dataset is overwritten. To avoid an accidental overwrite of the **Spatial Hierarchy** dataset, Siemens Digital Industries Software recommends that you restrict the authorization of performing **Spatial Hierarchy** production to select user accounts. The **VIS_mmvindexgen_admin_group** site preference is used to give the user group the authority to run the **mmvindexgen** translator from the **Teamcenter Translation** menu.

  - This translator requires Perl version 5.0.3 or later.

  - (Optional) To define one or more revision rules to generate the **Spatial Hierarchy** dataset, open the **mmvindexgen.config** file from the *Dispatcher_Root***Module\Translators\mmvindexgen** directory and edit the **revision_rules** option.

    By default, this option does not have any revision rule and generates **Spatial Hierarchy** in the unconfigured mode.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator.

The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

   For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

   > Note:
   >
   > For more information, see the **Readme** file in the **Translators***translator_name* directory.

   - Windows: *translator_name*.**bat -help**
   - Linux: *translator_name*.**sh -help**

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file.

   NxTransDirect translator example:

   ```
   "%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=Dispatcher-client-proxy-user
   -p=password
   -changeOwnerToCad=false "-pim=yes" %*
   ```

   If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

   After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

Enable PcbToFatf, PartUtility, and populateFSC translators

- **PcbToFatf** translator

  Prerequisites:

  Verify that the **TransExecutable dir** is correctly specified for this translator.

- **PartUtility** service

  Runs the NX Part utility and saves NX parts in the current version of NX.

  Prerequisites:

NX installed with **UGMANAGER** and **PVTRANS** modules.

- **populateFSC** translator

  Supports FSC caching of structured context object (SCO) content at non-database sites. The translator generates a PLM XML file of the SCO object, parses it for file global unique IDs (GUIDs), and generates read tickets for all the files referenced in the PLM XML file. You can use the read tickets to load the FSC cache of a target or a distant FSC of the same database.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

   For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

   > Note:
   >
   > For more information, see the **Readme** file in the **Translators***translator_name* directory.

   - Windows: *translator_name*.**bat -help**
   - Linux: *translator_name*.**sh -help**

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file.

NxTransDirect translator example:

```
"%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=Dispatcher-client-proxy-user
-p=password
-changeOwnerToCad=false "-pim=yes" %*
```

If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

Enable QSEARCH_process_queue, QueryScos, and RenderMgtTranslator translators

- **QSEARCH_process_queue** service generates spatial index boxes.

  Prerequisites:

  - Teamcenter and the **qsearchprocessqueue** utility should be installed to use this service.
  - This translator requires the Teamcenter password.

    To encrypt a password file, you set a temporary environment variable to the password you want to encrypt, and then generate an encrypted password file using the **-encryptpwf** argument for the install utility.

- **QueryScos** translator

  Generates a list of structure context objects (SCOs) for a given saved query or folder containing the SCOs. The generated SCOs are used by the **RdvContextDBDownload** or **RdvContextNONDBDownload** translators.

  Prerequisites:

  - Start the **start_sco_dispatcher** utility to invoke the **query_scos** translator.
  - Install the **RdvContextDBDownload** or **RdvContextNONDBDownload** translator.

- **RenderMgtTranslator** service translates files from one input file format to another output file format and stores the output files in Teamcenter.

  Prerequisites:

- Requires **PreviewService**.

  Use TEM or Deployment Center to install and configure **PreviewService**.

  Teamcenter Visualization Convert is a prerequisite for **PreviewService**.

  > Note:
  >
  > Verify whether Teamcenter Visualization Convert is installed correctly by performing a test conversion using the **prepare** command line utility. You can run this utility from the **VIS_***version***/VVCP** directory.

- You can use any other translator if the translator is defined as a dispatcher service configuration in Business Modeler IDE and the **ServiceToExeMap** attributes in the *Dispatcher_Root***\Module\conf\translator.xml** file are set to the location of the translator **.bat** (Windows) or **.sh** (Linux) files.

  > Note:
  >
  > You can use the **rendermgt_previewservice.bat** or **rendermgt_previewservice.sh** file in the *Dispatcher_Root***\Module\Translators\previewservice** directory as a template for creating the **.bat** or **.sh** file for your custom translator.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***\Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator***.bat** (Windows) or *translator***.sh** (Linux) file from the *Dispatcher_Root***\Module\Translators***translator_name* directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

> Note:
>
> For more information, see the **Readme** file in the **Translators***translator_name* directory.

- Windows: *translator_name***.bat -help**
- Linux: *translator_name***.sh -help**

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name***.bat** (Windows) or *translator_name***.sh** (Linux) file.

   NxTransDirect translator example:

   ```
   "%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=Dispatcher-client-proxy-user
   -p=password
   -changeOwnerToCad=false "-pim=yes" %*
   ```

   If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

   After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

Enable Simulation Process and Data Management translators

- The **Simpgen** translator is primarily used to create assembly-level simplification geometry.

  Prerequisites:

  - The **Simpgen** translator requires Perl 5.0.3 or later.
  - Licenses for Teamcenter and Teamcenter lifecycle visualization mockup (for the **JtOptimize** utility). An additional license is required to work with **JtSimplification** data in the viewer.

    For more information, contact your Siemens Digital Industries Software representative.
  - OpenGL on the Teamcenter server. OpenGL is an open source environment for developing 2D and 3D graphics applications.

    For more information, see the OpenGL website.

The **Simpgen** translator heavily leverages the graphics hardware. Siemens Digital Industries Software recommends that you install a high-end graphics card for better performance. When this translator is run on a Linux machine, a user with appropriate permissions must locally log on to the machine to initialize the graphics hardware and grant hardware permission (for example, using the **xhost** command) to other users who start the dispatcher remotely.

For more information, see the *Dispatcher_Root***Module\Translators\simpgen\Readme.txt** file.

- The clients and Dispatcher Server should point to the common staging directory and both should have read and write permissions to that directory. Additionally, the common staging directory should have sufficient disk space to store temporary **PLM/XML** and **JT** files created during the translation process.

  The Dispatcher Server allows translations through remote method invocation (RMI) mode. RMI mode requires a common staging directory.

- When **JtSimplification** production is performed on an assembly under which a **JtSimplification** dataset already exists for one or more item revisions, the existing **JtSimplification** dataset is overwritten. To avoid accidental overwrite of the **JtSimplification** dataset, Siemens Digital Industries Software recommends that only a select few user accounts are given the authority to perform **JtSimplification** production.

  VIS_simpgen_admin_group site preference is used to give the user group the authority to run the **Simpgen** translator from the Teamcenter **Translation** menu.

  The default value for this site preference is **SimpGenAdmin**. You can specify the name of a Teamcenter group as a valid value.

- The **SimProcess** service supports remote launch for Simulation Process and Data Management.

  Prerequisites:

  - The **tc_launch_sim_process_ts_pl.bat** file in the **module/SimProcess** directory points to the **Perl** location. Make sure that the **Perl** location is correct.

  - The clients and Dispatcher Server should point to the common staging directory and both should have read/write permissions to that directory.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

   For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

   > Note:
   >
   > For more information, see the **Readme** file in the **Translators***translator_name* directory.

   - Windows: *translator_name*.**bat -help**
   - Linux: *translator_name*.**sh -help**

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file.

   NxTransDirect translator example:

   ```
   "%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=Dispatcher-client-proxy-user
   -p=password
   -changeOwnerToCad=false "-pim=yes" %*
   ```

   If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

   After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

Enable the store_and_forward translator

The **store_and_forward** translator supports the store and forward functionality in batch mode. It executes the **move_volume_files** utility with a new **-transfersaf** argument. The utility queries the entire

database for the extent of user's files under local volumes and transfers them to their respective default volumes. The utility also creates and schedules a dispatcher task to delete the local volume files after the ticket expiry interval.

Default local volumes are temporary local volumes that allow files to be stored locally before they are automatically transferred to the final destination volume. This functionality is referred to as store and forward functionality.

Prerequisites

Ensure that the *store_and_forward.pl* file under *Dispatcher_Root**Module**Translators**store_and_forward* has correct values specified for the user name (**-u=**) and password file (**-pf=**) parameters.

If these parameters have values set as "**CHANGE_ME**", manually edit them, setting them to the correct values.

> Note:
>
> The password file specified using the **-pf** parameter should be a file that is encrypted using "**install -encryptpwf**".

Enable the translator

1. Set the **TC_Store_and_Forward** preference.

   You must set this preference to enable store and forward functionality.

2. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory and enable the **StoreAndForward** translator.

3. In the **Select Translators** panel in TEM or Deployment Center, select the **StoreAndForward** translator.

   You must specify this value while enabling this translator using TEM or Deployment Center.

   This is generally the same as the **Fms_bootStrap_Urls** preference value.

4. To set the **store_and_forward** translator as a repeat task, in My Teamcenter, choose **Translation→Translate**, select the translation service, and click **Next**. Select the **Repeating** option and click **Finish** to start the translation.

Note:

The **Repeating** option does not appear by default. You must set the **ETS.Repeating_UI.<ProviderName>.<ServiceName>** preference to **TRUE** to display the repeating tasks functionality.

To avoid unpredictable behavior, the (time) interval in repeating tasks must be greater than the translation time.

Enable Teamcenter Substance Compliance services

The Teamcenter Substance Compliance solution requires the following dispatcher services:

- *Compliance Results Validation*: Invalidates substance compliance results based on the expiry of exemptions on related objects.
- *Supplier Declaration Import*: Creates the material and substance information in Teamcenter based on material substance declaration (MSD) files.
- *Supplier Declaration Re-Request*: Sends an MSD request again to suppliers, for all qualified vendor parts.
- *Email Polling*: Polls email on a dedicated email account based on Teamcenter rules.
- *Generate outbound declaration*: Generates an outbound declaration file in IPC formats for the selected object.
- *Send reminders for declarations to vendors*: Creates reminders for the declarations in Teamcenter which are about to expire. It also sends reminders to vendors who have not responded to a declaration request earlier within the stipulated time.

Enable these dispatcher services as follows.

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.
2. To activate the **SubsCmplValidationService** service, set the **isactive** attribute to **true**.
3. To activate the **SubsCmplMSDImportService** service, set the **isactive** attribute to **true**.
4. To activate the **SubsCmplMSDRerequestService** service, set the **isactive** attribute to **true**.
5. To activate the **EmailPollingService** service, set the **isactive** attribute to **true**.

6. To activate the **SubscmplGenerateDeclarationService** service, set the **isactive** attribute to **true**.

7. To activate the **SubscmplDeclarationRemindersService** service, set the **isactive** attribute to **true**.

8. Edit the **CHANGE_ME** tags for all the services you have activated in the respective *service_name***.bat** (Windows) or *service_name***.sh** (Linux) files in the **\Module\Translators***service_name* directory.

   Refer to the respective files.

9. From the command prompt, run the service in standalone mode to verify whether it is working properly.

   To view help about running the translator in a standalone mode, from a command prompt, change to the *Dispatcher_Root***\Module\Translators***service_name* directory and call *service_name***.bat -help**.

Enable NX translators

- **NxClone** service

  Calls the **ug_clone.exe** utility to clone NX parts.

  NX must be installed with the UGManager module to use this service.

- **NxNastran** service

  Used to launch NX Nastran remotely using Simulation Process and Data Management.

  Prerequisites:

  - The clients and Dispatcher Server should point to the common staging directory and both should have read/write permissions to that directory.

  - To launch a service remotely, ensure that the service name in the **translator.xml** file matches the process name in the **Simulation Tools Configuration** window in CAE Manager.

- **NxTransDirect** service

  Translates NX files (**.prt**) to JT (**.jt**) and other formats and directly stores the JT files in Teamcenter.

  Create the JT file synchronously when you save a part.

  You can create a customer default that allows you to create the JT file synchronously when you save a part. It is applicable only when the **Save JT Data** option is also selected. If this customer default is

not selected, JT file creation occurs as a batch translation in the background, which reduces the overall save time.

In Teamcenter 11.4 or later, this customer default is not available. Instead, you can set the **NX_ETS_NXTRANSDIRECT_ENABLED** preference to **False**.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. (Only for **NxTransDirect** service) To specify user credentials explicitly for translation process, comment out the following properties from the **translator.xml** file. It is not recommended to add the password as plain text to the **translator.xml** file or *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file. Instead, generate an encrypted password file using the **-encryptpwf** argument with the Teamcenter install utility. For more information, see **step 5**.

   Example:

```
<NxTransDirect provider="SIEMENS" service="nxtransdirect"
isactive="false" OutputNeeded="false">
      <TransExecutable name="nxtransdirect.bat" dir="&MODULEBASE;/
Translators/nxtransdirect"/>
      <Options>
        <Option name="inputpath" string="-inputFile="
                    description="Full path to the input file."/>
        <!-- Comment out user name, password, group, encrypted password and
autologin options -->
        <!-- <Option name="clientoption" optionkey="user" string="-
u="
                                value=""
                                description="Specifies username for
NX Translator login to Teamcenter."/>
```

```
        <Option name="clientoption" optionkey="password" string="-p="
               value=""
               description="Specifies password for NX Translator
login to Teamcenter."/>
        <Option name="clientoption" optionkey="group" string="-g="
               value=""
               description="Specifies group for NX Translator login
to Teamcenter."/>
        <Option name="clientoption" optionkey="encrypted_password"
string="-encrypt="
               value="true"
               description="Specifies whether or not password is
encrypted for NX Translator login to Teamcenter."/>
        <Option name="clientoption" optionkey="use_module_user"
string="-autologin="
               value="true"
               description="Specifies whether  to auto login as
module user."/> -->
        <!-- End of commenting -->
        <Option name="clientoption" optionkey="storeInSourceVolume"
string="-storeInSourceVolume="
                              value="false"
               description="Specifies whether to store the
translated data in the same volume as source."/>
        <Option name="clientoption"
optionkey="updateExistingVisData" string="-updateExistingVisData="
                                     value="false"
               description="Specifies whether to update existing
translated data."/>
        <Option name="clientoption" optionkey="changeOwnerToCad"
string="-changeOwnerToCad="
                                              value="false"
               description="Specifies whether to change the
translated data to the same owner as source."/>
      </Options>
      <TransErrorStrings>
        <TransInputStream string="Cannot"/>
        <TransInputStream string="Error"/>
        <TransInputStream string="exception"/>
        <TransInputStream string="ERROR"/>
        <TransErrorStream string="Errored"/>
        <TransErrorStream string="failed"/>
      </TransErrorStrings>

   </NxTransDirect>
```

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file.

NxTransDirect translator example:

"%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=*Dispatcher-client-proxy-user* **-pf=***pf_filepath_filename* **-changeOwnerToCad=***false* **"-pim=***yes***" %***

Example: **-pf=***C:\Users\user_name\AppData\encrptPass.pwf*

For more information, see *Manage password files* and *install* utility in *Teamcenter Utilities*.

If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

6. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

   For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

   > Note:
   >
   > For more information, see the **Readme** file in the **Translators***translator_name* directory.

   - Windows: *translator_name***.bat -help**
   - Linux: *translator_name***.sh -help**

Enable tc3dpdftrans and tcmfg_update_productviews translators

- The **tc3dpdftrans** translator creates 3D PDF reports.
- The **tcmfg_update_productviews** translator updates product views in a batch mode.

Enable the translators

You can use TEM or Deployment Center to install the translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***\Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate this translator.

   > Note:
   >
   > By default, this attribute is set to **false**.

3. Open the *translator*.**bat** (Windows) or *translator*.**sh** (Linux) file from the *Dispatcher_Root***Module\Translators***translator_name** directory and edit the **CHANGE_ME** tags.

   For more information about **CHANGE_ME** tags, see the **.bat** file for the translator.

4. Run the translator in standalone mode from the command prompt to verify whether it is working properly.

   For more information about running the translator in standalone mode, from a command prompt, change to the **Dispatcher\Translators***translator_name* directory and run the appropriate command for your platform:

   > Note:
   >
   > For more information, see the **Readme** file in the **Translators***translator_name* directory.

   - Windows: *translator_name*.**bat -help**
   - Linux: *translator_name*.**sh -help**

5. (Optional) To set up *Dispatcher-client-proxy-user* as the owner of the translated files, make the following change to the *translator_name*.**bat** (Windows) or *translator_name*.**sh** (Linux) file.

   NxTransDirect translator example:

   ```
   "%UGII_BASE_DIR%\ugmanager\run_nxtrans.bat" -u=Dispatcher-client-proxy-user
   -p=password
   -changeOwnerToCad=false "-pim=yes" %*
   ```

   If you set up *Dispatcher-client-proxy-user* as the owner of the translated files, you must **create a dispatcher client access rule**.

   After installation, you must add a rule to the access rule tree permitting the translation service proxy user to update attributes of a **DispatcherRequest** object. If this access rule is not created correctly, the dispatcher client reports errors.

Enable tcftsindexercharacteristicsindex translator

The **tcftsindexercharacteristicsindex** translator adds characteristics information created using third-party tools such as **NX Inspector** to Teamcenter.

For more information, see *About characteristics* in *Model-Based Systems Engineering*.

Prerequisites

Parameter Management should be deployed in Teamcenter.

Enable the translators

You can use Deployment Center to install the **Characteristics Indexer** translator or manually edit the **translator.xml** file to enable the translator. The following is an example of how to edit the **translator.xml** file to enable the translator.

1. Open the **translator.xml** file from the *Dispatcher_Root***Module\conf** directory.

2. Set the **isactive** attribute to **true** to activate the **TcFtsIndexerCharacteristicsIndex** translator.

   Note:

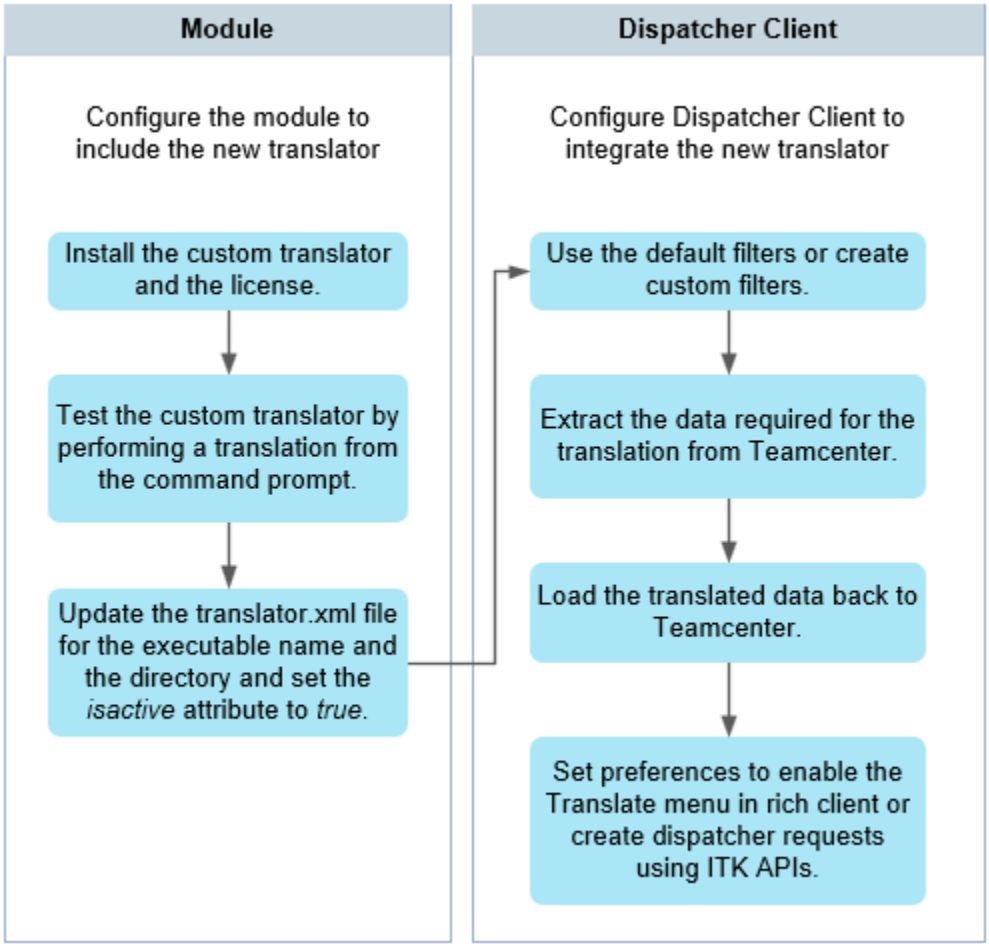
   By default, this attribute is set to **false**.

   When you install this translator using Deployment Center, the attribute is set to **true**.

7. Add custom translators to Dispatcher

How to add custom translators to Dispatcher

The following is a typical sequence of tasks for adding custom translators or new translators to Dispatcher.



Configure the module to include the new translator

- *Install the new translator and the license*

  Install the new translator on a machine or module as per your deployment strategy. Ensure that the new translator is properly licensed.

- *Test the custom translator by performing a translation from the command prompt*

  Perform a translation from the translator in standalone mode from the command prompt to verify whether it is working properly.

- *Update the translator.xml file to include the new translator*

  Update the translator.xml file for the executable name and the directory and set the **isactive** attribute to **true**.

Configure the Dispatcher client to integrate the new translator

- *Use the default filters or create custom filters*

  You can modify the **default validation rules** to ensure that the same primary object is not sent for translation again or modify **default filters** to exclude specific translators, exclude translation requests based on priority, and exclude translation requests based on owning user name.

  Alternatively, you can create custom filters by using the **sample Filter.java file** from the *DISPATCHER_ROOT***/DispatcherClient/sample/integration/filters** directory.

- *Extract the data required for the translation from Teamcenter*

  You can extract the data required for the translation by using the **TaskPrep Java class**.

- *Load the translated data back to Teamcenter*

  You can load the translated data back to Teamcenter by using the **DatabaseOperation Java class**.

- *Set preferences to enable the Translate menu in rich client or create dispatcher requests using ITK APIs*

  You add **Dispatcher preferences** for the translation options to appear in the **Translation Selection** dialog box in the rich client.

  Alternatively, you can create **dispatcher requests using ITK APIs**. The Dispatcher client provides server APIs to support customized triggering of translations. You can modify the sample **dispatcher_create_rqst_itk_main.c** source code or create a new ITK command line C source file.

Create custom filters

What are filter classes?

Filters are implemented by subclasses of the **RequestFilter** class. Filter instances are created by the **FilterFactory**[1] class utilizing methods of the **RequestFilter** class. All translation services apply filtering to the results returned from the query for dispatcher requests.

1 **FilterFactory** is a class supplied by Dispatcher. It is a mechanism used by the service classes for creating instances of **RequestFilter** subclasses—for example, **PriorityFilter**.

To create a new filter, create a new subclass of the **RequestFilter** class providing an implementation for each of the methods specific to the new filter.

A sample **Filter.java** file is provided in the *DISP_ROOT*/**DispatcherClient/sample/integration/filters** directory.

RequestFilter contains the **properties** data member. The **properties** data member is an object that contains the configuration required and optional properties and values that are established when the dispatcher client starts.

The **RequestFilter** contains the following methods:

- **getRequiredPropertyNames**

  This method is used by the **FilterFactory** class to return an array of property names of required configuration properties. These properties are those that must be specified by the filter configuration. If no value is specified for these properties, the **FilterFactory** class logs an error in the Extractor service instance log file and no filter instance is created.

  ```
  public abstract String[] getRequiredPropertyNames();
  ```

- **getOptionalPropertyNames**

  This method is used by the **FilterFactory** class to return an array of property names of optional configuration properties. These properties are optional.

  ```
  public abstract String[] getOptionalPropertyNames();
  ```

- **setProperties**

  This method is used by the **FilterFactory** class to allow a specific **RequestFilter** subclass access to property values. The subclass implementation must execute the **super** method.

  ```
  public void setProperties(Properties p) throws Exception
      {
          properties = p;
      }
  ```

- **filterRequest**

  This method is used to filter dispatcher request objects for processing by the dispatcher client. It is run by the dispatcher client to filter the results of the query for dispatcher request for processing.

  ```
  public abstract boolean filterRequest( IMANComponent request )
  throws Exception;
  ```

Develop custom Dispatcher client filters

To customize a dispatcher client filter in a Microsoft Windows environment:

> Note:
>
> Adapt the procedure for Linux as required.

1. Create the custom dispatcher client filter Java source file. Use a custom package designation.

   A sample **Filter.java** file is provided in the *DISP_ROOT***/DispatcherClient/sample/integration/filters** directory.

2. Ensure that the appropriate Java SDK is installed and Java path information is correctly set.

3. Compile the Java source file using the following command:

   ```
   javac -classpath    Add jars created in the classpath of runDispatcher.bat/sh file
   myFilter.java
   ```

4. Archive the new Java classes. These new classes, along with other customized dispatcher client classes, are combined into the CAD Manager JAR file or a single site-specific custom JAR file.

   ```
   jar cf jar-file-name.jar myFile.class myFile2.class …
   ```

5. Copy the new JAR file to the *DISP_ROOT***/DispatcherClient/lib** directory.

6. Modify the *DISP_ROOT***/conf/Service.properties** file to specify the new filter and associated properties.

   Add the filter in the **FILTERS** section of the **Service.properties** file.

7. Run the dispatcher client and verify the operation of the filter.

Extract data and load data required for translation

What is the extract-transform-load (ETL) model?

The dispatcher client employs the ETL model (Extract-Transform-Load) for data translation. It provides a framework that supports extraction of data for translation, translation, and loading of translation results. The dispatcher client is implemented as a standalone SOA application.

- Extraction provides a collection of input data to be submitted as a translation task to the Dispatcher Server. The nature of the extracted data is dependent upon the input required by the desired translation.

- After extraction, the dispatcher client submits a translation task to the Dispatcher Server. The Dispatcher Server then processes the translation tasks.

- After the translation is complete, the dispatcher client loads the translated data in the appropriate Teamcenter target data model.

  The nature of the translation results is dependent on the output generated by the desired translation. Storage of the results is also dependent on the target data model.

The dispatcher client provides support for translator integrations and related data models with two Java classes: one for preparation of translator input (**TaskPrep** class) and one for storage of translator output (**DatabaseOperation** class). These two classes provide a structure for translator- and data model-specific logic implementation. The dispatcher client configuration properties define the mapping between translators and supporting classes.

These classes provide a structure for translator-specific and data model–specific logic implementation. The dispatcher client configuration properties define the mapping between translators and supporting classes.

Extract and load customization

If you want to translate a single dataset and attach a single result dataset back using preferences, see *Basic TaskPrep and DatabaseOperation Class* section in this topic. However, if you have special cases to handle, see *Extract data*, *Load data*, and *Compile source files and configure the translator* sections in this topic.

> Note:
>
> Adapt the procedures for Linux as required.

1. Determine the following input requirements of the translator:

   - Data to be translated.

   - Datasets containing the data to be translated.

   - Named references are required to perform the translation.

2. Decide the output of the translator.

3. Determine the Teamcenter data model under which the result files are stored. Utilize the Teamcenter standard data model and supported extensions. Be sure the data model is supported by any downstream tools that are to consume translation results from the Teamcenter database.

For more information about packages and classes, see *Dispatcher_Root***DispatcherClient\docs\dc\javadoc\index.html**

Extract data

1. Design the logic of the new **TaskPrep** class **prepareTask** method to satisfy translator input, and other Dispatcher requirements. Design the necessary helper classes. Sample **TaskPrep** classes and property files are provided in the *Dispatcher_Root*/**DispatcherClient/sample/integration/translator** directory.

2. Extend the dispatcher client **DefaultTaskPrep** abstract class in a new **TaskPrep** class in a package specific to the translator.

   For example: *your path*.**translator.***provider name*.*translator name*

3. **Compile source files and configure the translator**.

Load data

1. Determine the **DatabaseOperation** method implementations that are necessary to load the translator results in the Teamcenter database. This includes the **load** method at a minimum.

2. Design the logic of the required **DatabaseOperation** methods and any necessary helper classes. Sample **DatabaseOperation** classes and property files are provided in the *DISP_ROOT*/**DispatcherClient/sample/integration/translator** directory.

3. Implement the new **DatabaseOperation** class in the same package with the **TaskPrep** class.

4. **Compile source files and configure the translator**.

Compile source files and configure the translator

1. Implement the associated helper classes.

2. Compile the Java source files.

   `javac -classpath` *Classpath created by the runDispatcherClient.bat/sh file* `myFile.java`

   Typically, you would add all the JARs in the **DispatcherClient\lib** directory to the classpath.

3. Create the **TS***Translator name***Service.properties** file to incorporate the new translator integration classes.

   The *provider* and *service* attribute values specified by the module translator configuration in the **translator.xml** file must be used in constructing the property names (keys).

   Edit the **Service.properties** file to import the new **TS***Translator name***Service.properties** file.

4. Archive the new Java classes and the **TS***Translator name***Service.properties** file. These new classes along with other customized dispatcher client classes are most simply combined into a single application or custom JAR file.

   Make sure the **TS***Translator name***Service.properties** file is in the root directory.

   ```
   jar cf jar file name.jar myFile.classmyFile2.class TSTranslator
   nameService.properties
   ```

5. If the JAR file contains only translator integration classes, copy the new JAR file to the *DISP_ROOT***I DispatcherClient/lib** directory.

6. To see all preferences, refer the *Dispatcher_Root***\DispatcherClient\install\ets_env.xml** file and update them based on your requirements. The preference description contains the details. **ETS_trans_rqst_referenced_dataset_types** is one of the preferences. If this file is modified, import it using the **preferences_manager** utility.

   > Example:
   >
   > Modify the **ETS_trans_rqst_referenced_dataset_types** site preference to add the application or customized dataset type name of the datasets that may be specified as a primary objects in the dispatcher request for this translator. Addition of a dataset type to this preference allows cancellation of a dispatcher request and subsequent deletion of the primary objects while the translation is *in process*.

7. Start the dispatcher client.

8. Trigger a dispatcher request and verify that the new **TaskPrep** class run by the dispatcher client processes the dispatcher request correctly. Examine the contents of the Dispatcher Server file server task directory (or task staging location if configured for shared staging location). Verify that the XML task file is present and that its contents are correct. Verify that all the expected translator input files are present.

9. Verify that the translation task is successfully submitted to the Dispatcher Server scheduler.

10. When translation completes, verify the contents of the Dispatcher Server task directory (or task staging location) to see that all expected result files are present.

11. Verify that the new **DatabaseOperation** class runs by the dispatcher client processes the dispatcher request and correctly loads the result files. Confirm that the location and ownership of the stored results is correct.

Basic TaskPrep and DatabaseOperation Class

1. Determine the preferences required by the new **DatabaseOperation** class and add them to the **TaskPrep** class.

Also look at the common **tozipfile** translator preferences to support MSWord and MSWordX datasets. For examples, refer the *Dispatcher_Root***DispatcherClient\install\basic_env.xml** file.

2. Implement the new translator site preference XML file and use the **preferences_manager** executable to import the file.

   preferences_manager -u=*user_name* **-p=***password* **-g=***dba* **-mode=***import* **-scope=***SITE* **-action=***override* **-context=***Teamcenter* **-file=***your_XML_file*

For more information about how to create a basic example of extract and load in Dispatcher Client, see *Readme* in the *Dispatcher_Root***DispatcherClient\sample\integration\translators\ppttopdf** directory.

Integrate new translators in the rich client

The following procedure explains how to integrate new translators in the Teamcenter rich client and to add them to the **Translate** menu.

Refer to **Extract and load customization** before you completing the procedures in this topic.

1. Determine the preferences required by the RAC UI and add them to the *Translator name***_env.xml** file. Keep in mind that some preferences may be required by the **TranslateMenu** classes and utilize these preferences before creating new ones.

   For more information, refer the *START SINGLE SELECTION PREFERENCES* section in *Dispatcher_Root***DispatcherClient\install\basic_env.xml** file.

   The *Translator-name***_env.xml** file should be in the *DISP_ROOT***/DispatcherClient/install** directory.

2. Implement the new translator site preference XML file and use the **preferences_manager** executable to import the file.

   preferences_manager -u=*user_name* **-p=***password* **-g=***dba* **-mode=***import* **-scope=***SITE* **-action=***override* **-context=***Teamcenter* **-file=***your_XML_file*

3. Confirm that the new translator shows up in the **Translate** menu when the source dataset is selected for translation.

For more information about how to create a dispatcher request for a new translator in rich client, see *Readme* in the *Dispatcher_Root***DispatcherClient\sample\integration\translators\ppttopdf** directory.

Dispatcher request object attributes

The Dispatcher task execution process starts by the creation of a **DispatcherRequest** object. The **DispatcherRequest** object is represented by the **DispatcherRequest** class.

Attribute	Description
Current state	Displays the current state of the dispatcher request.
Provider name	Displays the name of the provider of the translator.
Service name	Displays the name of the dispatcher service to be used.
Priority	Specifies a scheduling priority. Set it to low, medium, or high.
Task ID	Displays the task identifier assigned during the preparation process.
Creation date	Displays the date and time when the **DispatcherRequest** object was created.
Primary objects	Specifies one or more primary objects containing the data to be translated. When translating CAD data, primary objects are typically datasets.
Secondary objects	Specifies one or more supporting objects for the translation. Note: Secondary objects are optional and dependent on the requirements of the specific translation. When translating CAD data, secondary objects are typically item revisions.
Owning user	Displays the name of the user that created the request.
Owning group	Displays the name of the group that created the request.
History states	Displays various states that make up the history of the dispatcher request.
History dates	Displays timestamps associated with state values.
Type	Shows an application the purpose of the dispatcher request.
Mode	Represents a user-defined mode of operation.
Argument keys/argument data	Specifies one or more strings containing key/value pairs.
Data files keys/data files	Specifies one or more strings containing name/value pairs.
Start time	Stores the expected start time of the request.
Interval	Stores the number of seconds until the dispatcher request is translated again.
End time	Stores the end time of the request.

Dispatcher request states

The following information describes the various Dispatcher states associated with the translation process.

State	Description
INITIAL	Indicates the initial state of a newly created **DispatcherRequest** object.
PREPARING	Indicates that the data is being extracted from Teamcenter and that a Dispatcher Server task is being prepared.
SUPERSEDING	Supports the situation in which an original request contains a set of objects that require more than one translation—for example, if the set contains both CATIA V4 and CATIA V5 parts. This state indicates that successor dispatcher requests are being created.
SCHEDULING	Indicates that the task is being added to the Dispatcher Server scheduling queue.
TRANSLATING	Indicates the task is being processed by the Dispatcher Server.
LOADING	Indicates the task results are being loaded into Teamcenter.
COMPLETE	Indicates the Dispatcher task has successfully completed.
TERMINAL	Indicates the Dispatcher tasks has failed.
DUPLICATE	Indicates the primary objects (datasets) specified in the **DispatcherRequest** are also specified by one or more **DispatcherRequest** objects currently being translated.
DELETE	Indicates that the dispatcher object has been marked for deletion.
CANCELLED	Indicates that the dispatcher request has been canceled.
SUPERSEDED	Indicates a successful end state in which the original dispatcher request is replaced by one or more newer requests.
NO_TRANS	Indicates a successful end state in which the original dispatcher request contained objects that did not require translation.

Each dispatcher request object has an *initial* state value, one or more *working* state values, and one or more *end* state values. Each dispatcher service processes **DispatcherRequest** objects whose current state matches the *initial* state of the dispatcher service. When processing by the dispatcher service completes, it sets the state of the dispatcher request to an *end* state value.

Dispatcher request states	Value
Initial	**INITIAL**
In Progress	**PREPARING**, **SCHEDULED**, **TRANSLATING**, **LOADING**, **SUPERSEDING**
Final	**COMPLETE**, **DUPLICATE**, **DELETE**, **CANCELLED**, **SUPERSEDED**, **NO_TRANS**, **TERMINAL**

Translation process

1. Based on actions such as checkin and workflow, one or more **DispatcherRequest** objects are created. (In Teamcenter, the **DispatcherRequest** object is represented by the **DispatcherRequest**

class). The dispatcher requests are added to the **DispatcherRequest** database table with a state value of **INITIAL**.

2. Each dispatcher client queries the **DispatcherRequest** database table at the configuration-defined interval. The dispatcher client queries for **DispatcherRequest** objects with a state value of **INITIAL**.

3. The dispatcher client changes the **DispatcherRequest** state value to **PREPARING** to prevent other dispatcher client instances from operating on the same **DispatcherRequest** object.

4. The dispatcher request is validated to check whether it is a duplicate of another *in-process* request. If it is, the **DispatcherRequest** state value is set to **DUPLICATE**.

5. The translator-specific task preparation logic is activated to prepare data for translation input and create a task XML file to facilitate submission of the translation task to the scheduler. The input and XML files are collected in a task staging location. After task preparation is complete, the dispatcher client submits the task to the scheduler and sets the **DispatcherRequest** object state value to **SCHEDULED**. If an error occurs during preparation, the **DispatcherRequest** state value is set to **TERMINAL**.

6. When translation begins, the Dispatcher Server notifies the scheduler and the **DispatcherRequest** state is set to **TRANSLATING**.

7. When translation is successfully completed, the Dispatcher Server again notifies the dispatcher client and the **DispatcherRequest** state value is set to **LOADING**. If an error occurs or the translation task fails, the Dispatcher Server notifies the scheduler and sets the **DispatcherRequest** state value to **TERMINAL**.

8. After the result files are loaded to the task staging location, the result files are mapped to Teamcenter application data. The dispatcher client then activates the appropriate translator integration logic to store the translation results in the Teamcenter database containing the associated Teamcenter application data model. When the results are loaded, the dispatcher client sets the **DispatcherRequest** state value to **COMPLETE**. If an error occurs during loading, the **DispatcherRequest** state value is set to **TERMINAL**.

   If the translation fails, the request is updated with a log file in the loader. This is based on configuration settings.

Create dispatcher requests

Why create dispatcher requests?

You can customize dispatcher request creation by:

- Adding translator preferences to the Teamcenter database and then using the **Translate** menu commands for translations.

- Using APIs for creating dispatcher requests. You can use these APIs for batch processing of dispatcher requests or creating a dispatcher request from a workflow.

Create dispatcher requests in Teamcenter rich client

The Dispatcher client provides a single dispatcher request creation menu that is driven from preferences defined within Teamcenter. This event is activated by selecting the **Translation→Translate** menu command in My Teamcenter through one of the specific translator menu selections.

To add the translator to the **Translate** menu command, you must add translation preferences to Teamcenter.

If this menu command does not allow the creation mechanism you need in the rich client, a method is exposed to create a dispatcher request on the **com.teamcenter.rac.ets** plug-in within the rich client. The package, class, and method to create the request are:

Package: **external**

Class: **DispatcherRequestFactory**

Method: **createDispatcherRequest**

Create Dispatcher requests using ITK APIs

The Dispatcher client provides server APIs to support customized triggering of translations. The dispatcher client framework for creation of dispatcher requests allows customizations for batch translations and workflow.

You can create dispatcher request objects using the **DISPATCHER_create_request** API.

These API support the creation of dispatcher request object in a server environment.

You must include the **dispatcher_itk.h** file to provide the prototype for these Dispatcher API functions, and you must modify link scripts to link against the **libdispatcher** shared object.

Ensure that you specify primary and secondary objects as input to this API.

1. Modify the sample **dispatcher_create_rqst_itk_main.c** source code or create a new ITK command line C source file.

2. Compile the source module using the sample script provided in the **TC_ROOT/sample** directory by entering the following command:

```
compile -DIPLIB=none dispatcher_create_
rqst_itk_main.c
```

3. Copy the sample ITK link script provided in the **TC_ROOT/sample** directory. Edit the copied ITK link script to add the Dispatcher library, **libdispatcher.lib**, to the **link** command.

4. Run the modified ITK link script by entering the following command:

```
myLinkitk -o dispatcher_create_rqst_itk_main
dispatcher_create_rqst_itk_main.obj
```

5. Test the new ITK executable.

Creating Dispatcher requests using Teamcenter Services

Dispatcher provides a service to allow the creation of a dispatcher request using the service-oriented architecture within Teamcenter.

Batch translations

Existing non-Dispatcher customer translation processes typically employ a daily process that runs during off hours to examine the contents of the customers Teamcenter database and create the necessary visualization data. You can integrate this process with Dispatcher using Dispatcher Integration Toolkit (ITK) APIs.

For performing batch translations, the following are provided:

- ITK functions for use in custom batch programs.
- Command line ITK executable (with sample source) for use by batch scripts.

Dispatcher provides the **DISPATCHER_create_request** server ITK API. This ITK API supports the creation of dispatcher request objects in a server environment.

Workflow translations

No workflow action handlers or triggers are currently provided by Dispatcher. Because workflow is a customer process, it is intended that on-site customizations of workflow action handlers use the Dispatcher Integration Toolkit (ITK) API to create dispatcher requests as they are needed.

Add customizations during the Dispatcher client startup

To call a custom code during the Dispatcher client startup, do the following:

1. Create a custom class that extends **com.teamcenter.ets.ServiceInit** and implement the **init** method.

   Example:

```
{
public void init()
  {
    //call custom code.
  }
}
```

2. Add the compiled class to the **DispatcherClient\lib** directory as a jar file.

3. Add the class name to **DispatcherClient\conf\Service.properties** as follows:

```
#Custom class to invoke the methods that need to be called during
DispatcherClient startup.
#This class needs to implement the "init" method defined in
com.teamcenter.ets.ServiceInitclass.ServiService.InitClass=
com.xyz.CustomInit
```

Translation menu integration

The Translation menu

The **Translation** menu item appears on the menu bar, if you select the **Dispatcher Client for Rich Client** option during installation. The following procedures describe how to create the new menu items in the **Translation** menu bar and to start a new dispatcher request using the new menu item.

- Create a new menu item
- Create a dispatcher request using a new menu item

Create a new menu item

- Open **plugin.xml** in an Eclipse plugin project. In the **menuContribution** tag, add the following location;

```
<menuContribution
locationURI="menu:com.teamcenter.rac.ets.external.translate?
after=additions">
</menuContribution>
```

> Note:
>
> To place your new menu item after the **Translate...** menu item, use **after=additions** in the **menuContribution** tag.

To place your new menu item before the **Translate...** menu item, use **after=bottom** in the **menuContribution** tag.

Create a dispatcher request using a new menu item

- To create a new dispatcher request from a newly created menu item, use the **createDispatcherRequest** method as an exported function from the **DispatcherRequestFactory** class. This method allows you to create a dispatcher request easily from within your own plug-in by just referencing the main Dispatcher plug-in from Eclipse.

```
/**
     * Creates the DispatcherRequest.
     *
     * This function allows customers who wish to create a DispatcherRequest
     * through a menu or other plug-in within Teamcenter.
     *
     * [R] = required, [O] = optional
     *
     * @param [R] provider - string representing the name of the translator
     *                       provider
     * @param [R] service - string representing the name of the translator
     *                      to use
     * @param [R] priority - integer representing the priority used in
     *                       processing the request
     * @param [O] primaryObjects - array of components selected within
     *                             Teamcenter
     * @param [O] secondaryObjects - array of parent components for the
     *                               primary components
     * @param [O] startTime - time to start the request
     * @param [O] interval - number of times to repeat request
     * @param [O] endTime - time at which to stop repeating tasks
     * @param [O] type - string for user to define user specific
       type of request
     * @param [O] requestArgs - name/value arguments to pass to service
     *
     * @return true, if create DispatcherRequest
     *
     * @throws TCException the TC exception
     *
     * @published
     */

    boolean createDispatcherRequest(String provider, String service,
    int priority,
    TCComponent[] primaryObjects,
    TCComponent[] secondaryObjects,
```

```
String startTime, int interval, String endTime,
String type, Map<String, String> requestArgs) throws Exception
```

Customizing Translation Selection dialog box behavior

Why customize the Translation Selection dialog box behavior?

You can customize the **Translation Selection** dialog box behavior to do the following:

- Validate data.
- Dynamically add translation argument values to the dispatcher request.

You must perform the following steps to customize the **Translation Selection** dialog box.

- Create a custom plugin.
- Create a custom class.

Create a custom plugin

1. Create a new plugin based on the **com.teamcenter.rac.ets.translator.singleselection** plugin and extend the **requestCustomize** point.

2. You must provide the following information in the **plugin.xml** file.

   - **providerName**=*Name of the provider whose behavior needs to be customized*
   - **serviceName**=*Name of the service whose behavior needs to be customized*
   - **class**=*Name of the class that implements the customization*

   Your **plugin.xml** file looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
  <extension point="com.teamcenter.rac.ets.translator.
singleselection.requestCustomize">
    <translator class=" com.foo.customization.abctoxyz.
AbcToXyzRequestCustomization"
      providerName="SIEMENS" serviceName="abctoxyz">
    </translator>
  </extension>
</plugin>
```

You can add multiple translator tags to provide customization classes for different services, and the same plugin can be used to provide customization classes for different services. However, there can be only one customization class for one service and provider combination.

Create a custom class

1. Create the **com.teamcenter.rac.dispatcher.translator.singleselection.customization.***translator service name.New class name***RequestCustomization** class, which implements **com.teamcenter.rac.ets.translator.singleselection.customization.**

IRequestCustomizable.

2. This class must implement the following methods:

   - **validateRequestData**
   - **getValuesForKeys**

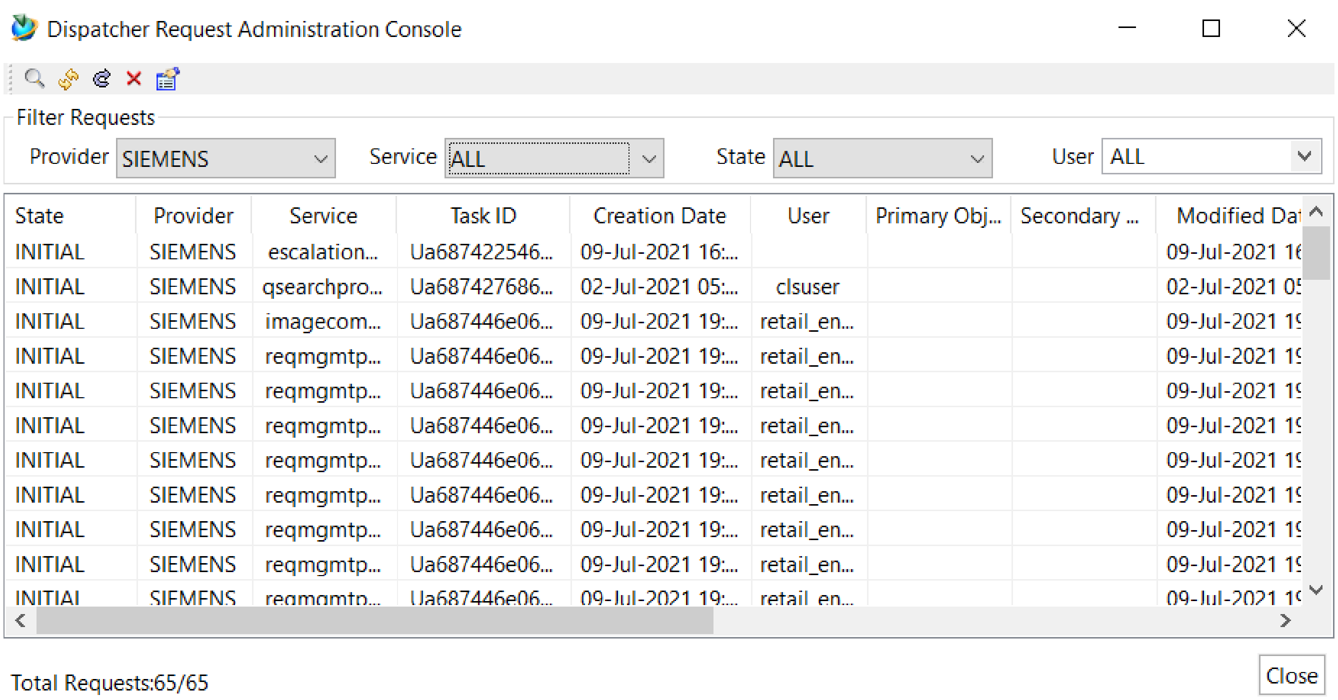
For more information about the extension point, refer to the **requestCustomize.exsd** schema located in the **com.teamcenter.rac.ets.translator.singleselection** plugin.

For an example on how to extend the **requestCustomize** extension point, see the **com.teamcenter.rac.dispatcher.customization** plugin.

8. Monitor and administer translation requests

Dispatcher request administration console interface

You can monitor and administer translation requests using the Dispatcher request administration console. To access the Dispatcher request administration console, in My Teamcenter choose **Translation→Administrator Console - All**.



Dispatcher request administration console buttons

Buttons	Description
	Refreshes all dispatcher requests.
	Refreshes the selected dispatcher request.
	Resubmits a dispatcher request for processing.
	Deletes the selected dispatcher request.
	Launches the properties dialog box for the selected dispatcher request.

Refresh, resubmit, delete, or filter Dispatcher requests

- Refresh dispatcher requests
- Submit a dispatcher request for processing again
- Delete dispatcher requests
- Filter dispatcher requests

Refresh dispatcher requests

- In the **Dispatcher request administration console**, click one of the following:
  - **Refresh All Requests (SHIFT+F5)** button to refresh all dispatcher requests.
  - **Refresh Request (F5)** button to refresh the selected dispatcher request.

Submit a dispatcher request for processing again

1. Select a dispatcher request.
2. Click the **Resubmit Request for Processing** button .

> Note:
>
> You can submit dispatcher requests again only in final states.

Delete dispatcher requests

1. Select a dispatcher request.
2. Click the **Delete Request** button .
3. To delete dispatcher requests which are the in-progress states, press the Shift key and click the **Delete Request** button .

> Note:
>
> The translation fails when you delete dispatcher requests which are in the in-progress states.

Filter dispatcher requests

1. From the **Filter Requests** section, select the filters from the available lists.

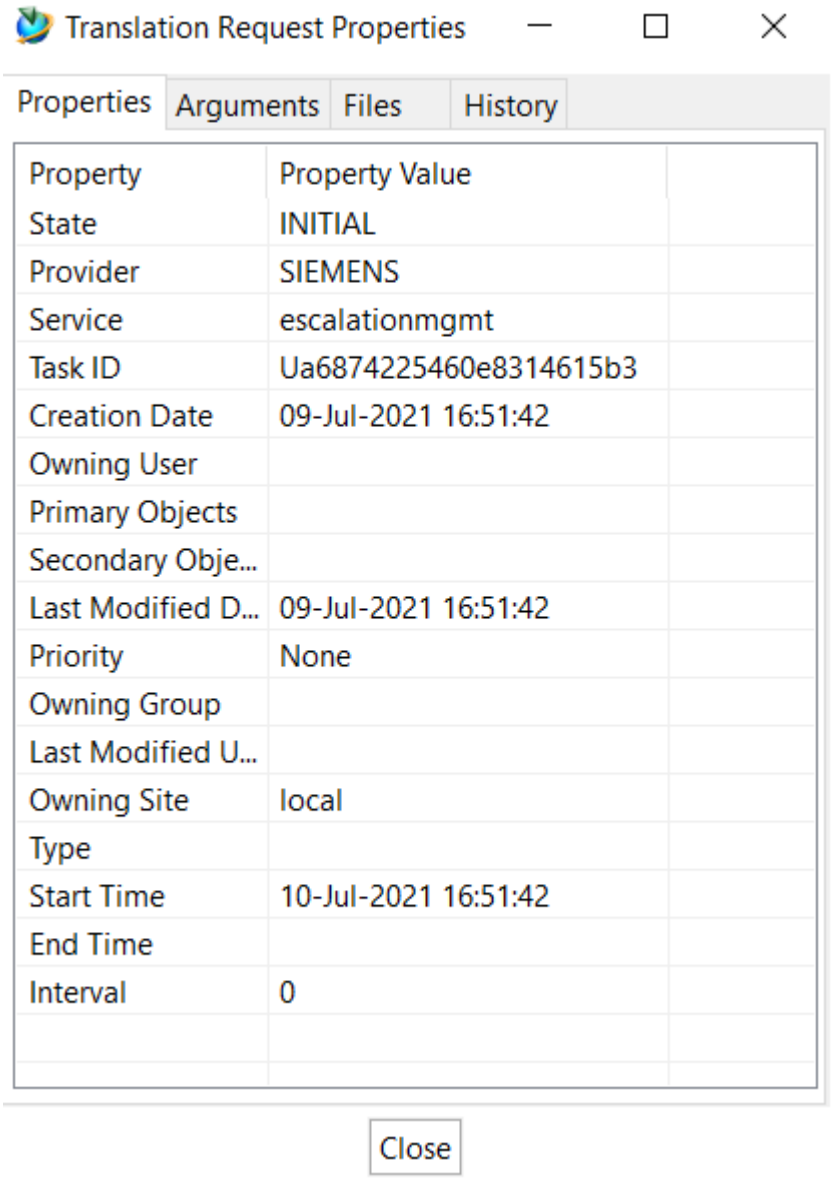
2. Filter dispatcher requests by selecting options provided in the following lists:

   - **Provider**
   - **Service**
   - **State**
   - **User**

View dispatcher properties

1. Choose a translation request and click the **Request Properties** button  .

2. The **Translation Request Properties** dialog box appears.



a. Click the **Properties** tab to view the properties of the selected dispatcher.

b. Click the **Arguments** tab to view the dispatcher arguments.

c. Click the **Files** tab to view the dispatcher log. You can download the dispatcher log to view it.

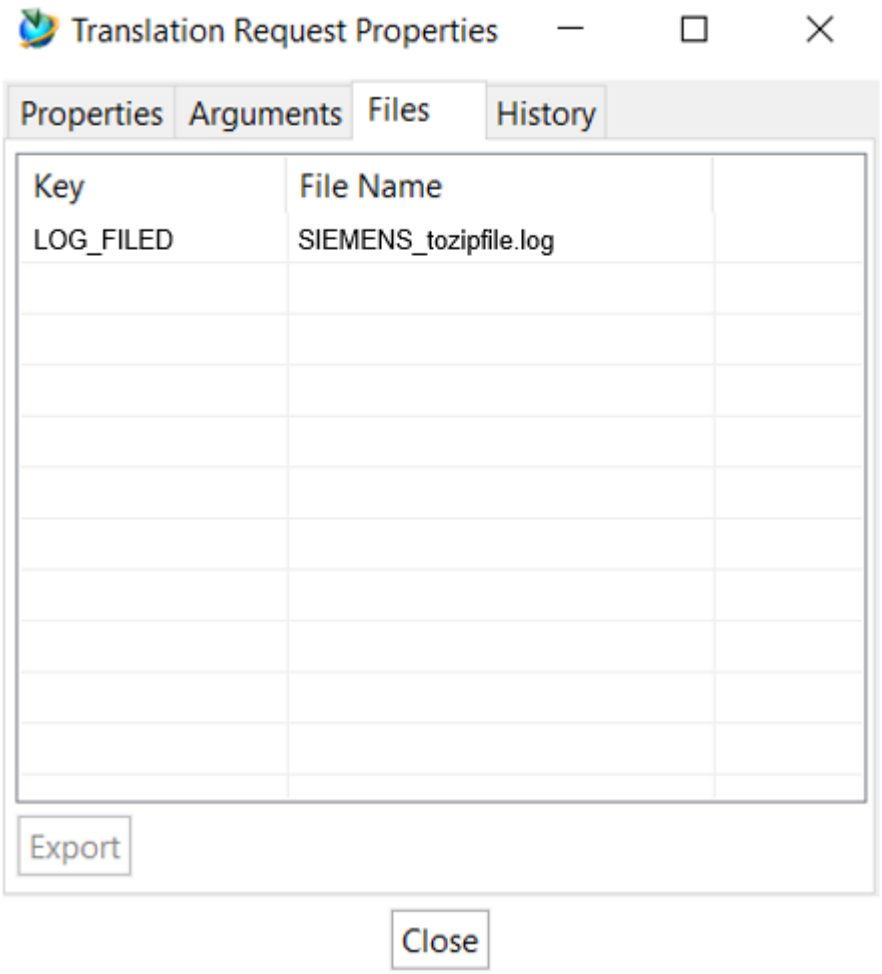
d. Click **History** to view a log of the different dispatcher states.

View logs attached to dispatcher requests

1. Choose a dispatcher request and click the **Request Properties** button .

2. In the **Translation Request Properties** dialog box, click the **Files** tab.

3. In the **Files** pane, select the log and click **Export** to view the log file locally.



9. Customize the rich client for Dispatcher

Enable translation options in the user interface by adding Dispatcher preferences

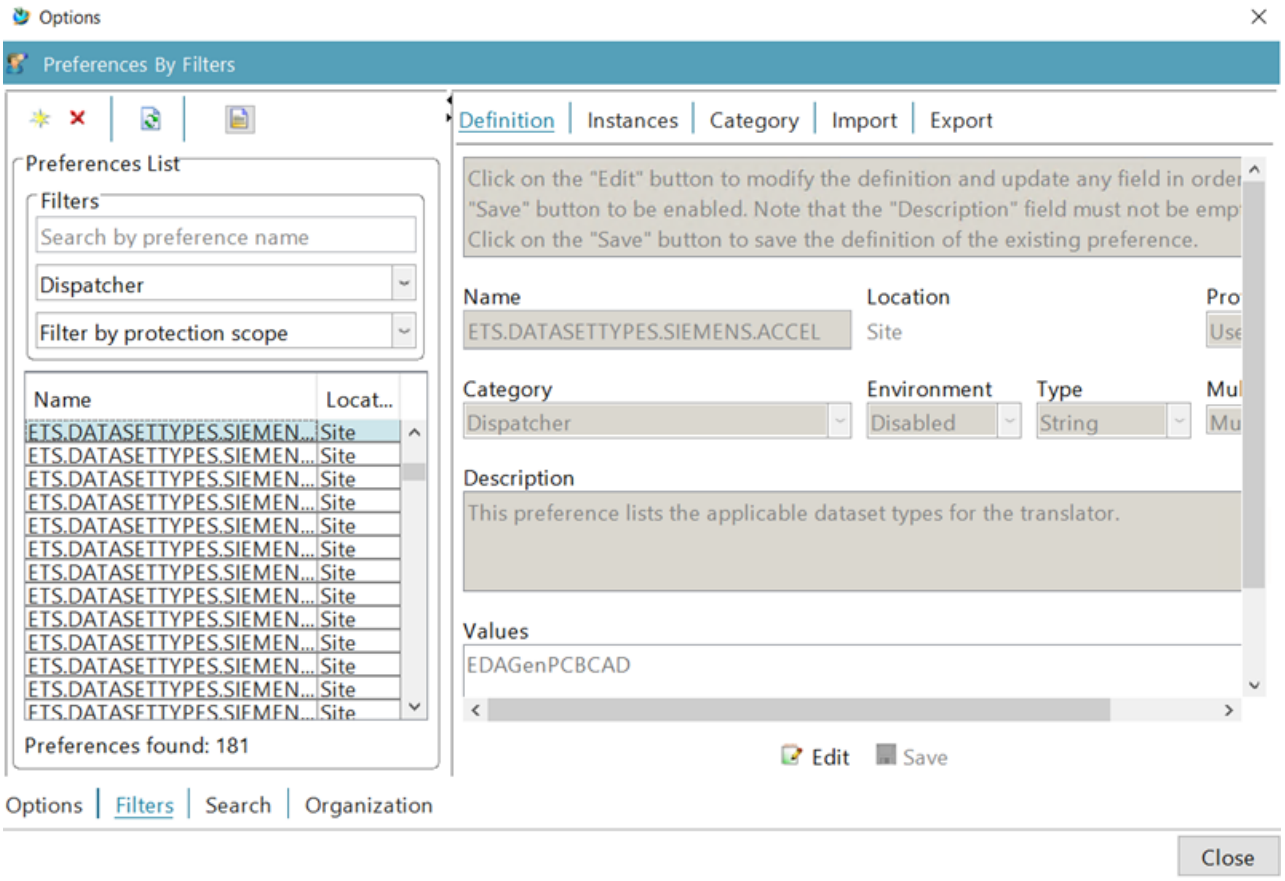
For translation options to appear in the **Translation Selection** dialog box, you must add Dispatcher preferences.

This dialog box allows you to select the translation provider and service for a selected data type. You can view the **Translation Selection** dialog box by choosing the **Translation→Translate** menu option in the My Teamcenter menu.

> Note:
>
> The translator preferences provided by Dispatcher are imported in Teamcenter during installation. You only need to add preferences of new translators or providers that you want to add to Teamcenter.

1. In My Teamcenter, choose **Edit→Options**.

2. In the **Options** dialog box, select **Filter**. This option is located at the bottom left corner of the **Options** dialog box.



3. In the **Preferences By Filter** dialog box, you can either modify the values of the existing preferences or add new preferences. To create new Dispatcher preferences, click the **Definition** tab and click **Create a new preference definition** in the **Preferences By Filter** dialog box.

   To modify the values of an existing preference, select the preference and change the values in the **Values** box.

   > Note:
   >
   > Access to preferences and preference hierarchy are specified by protection scope.

   You must set the following Dispatcher preferences:

Preference name	Description
ETS.PROVIDERS	Specifies the company that provides translators.
ETS.TRANSLATORS. *PROVIDER*	Specifies a list of translators for a specific provider.
ETS.DATASETTYPES. *PROVIDER.TRANSLATOR*	Specifies valid datasets for the selected translators.

Preference name	Description
	Note: Only the datasets specified in this preference are valid for translation.
ETS.PRIORITY. *PROVIDER.TRANSLATOR*	Specifies the priority to be assigned to a dispatcher request when the translator specified in this preference is used.
ETS.TRANSLATOR_ARGS. *PROVIDER.TRANSLATOR*	Specifies the list of translation arguments to be passed to the translator.

Note:

The order of values in the **Values** box determines the order in which they appear in the **Translation options** dialog box.

4. Restart the Teamcenter server and rich client.

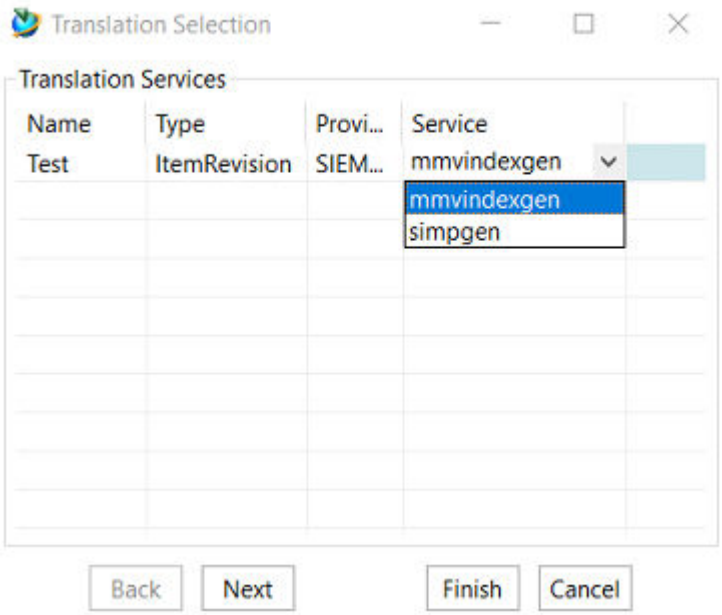
5. Select a dataset for which you defined preferences, and choose **Translation→Translate**.

   The translators defined for the dataset appear.

   Note:

   If there is more than one translator definition, all the translators appear in a list.

   The list with the translator values appears only if you select the cell that contains the translator service.



Create a Dispatcher request and extract the data required for the translation using sample preferences

You can use the **CreateRequestInRAC.xml** and **ExtractAndLoad.xml** sample preference files to create a dispatcher request for a custom translator in the rich client and to extract and load the data required for the translation in the Dispatcher client. These files are available in the *Dispatcher_Root***DispatcherClient\\sample\integration\translators\ppttopdf** directory. For more information, see the *Readme* file in the same directory.

Customizing the Dispatcher request administration console

Why customize the Dispatcher request administration console?

You can customize the Dispatcher request administration console to:

- Filter requests in the Dispatcher request administration console.

  For example, this can be done to filter out other site requests or filter out requests owned by other users.

- Create a new submenu under the **Translation** menu.

  This menu launches the Dispatcher request administration console with predefined filtered results.

  For example: Create a submenu called *Terminal only* to show only terminal requests in the Dispatcher request administration console.

Filter requests in the Dispatcher request administration console

1. Create a new plugin based on the **com.teamcenter.rac.ets** plugin and extend the **DispatcherAdminConsoleFilter** point.

2. You must provide the following information in the **plugin.xml** file.

   class=com.teamcenter.rac.dispatcher.sampledispaccustomization.handlers.NonTerminalFilter

   Class is the actual class provided by new plugin and it implements methods defined in the **com.teamcenter.rac.ets.customization.IDispatcherACCustomizable** interface.

   Your plugin.xml file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>
  <extension
    id="adminconsolenonterminal"
    point="com.teamcenter.rac.ets.DispatcherAdminConsoleFilter">
    <filter
      class="com.teamcenter.rac.dispatcher.sampledispaccustomization.
             handlers.NonTerminalFilter">
    </filter>
  </extension>
</plugin>
```

   There can be only one customization class for one service and provider combination. If there are multiple customizations, the Dispatcher request administration console ignores all the customizations and displays all the queried requests to user.

3. Create a class **com.teamcenter.rac.dispatcher.sampledispaccustomization.handlers.NonTerminalFilter** which implements **com.teamcenter.rac.ets.customization.IDispatcherACCustomizable**.

4. This class must implement the method in **IDispatcherACCustomizable**.

For more information about the extension point, refer to the **com.teamcenter.rac.ets.DispatcherAdminConsoleFilter.exsd** schema in **com.teamcenter.rac.ets** plugin.

Create a new submenu under the Translation menu

1. Create a new plugin based on the **com.teamcenter.rac.ets** plugin.

2. In the new plugin, create a new menu and create a handler class to handle the menu click.

3. Create a class named **com.teamcenter.rac.dispatcher.sampledispaccustomization.handlers.NonTerminalFilter** that implements **com.teamcenter.rac.ets.customization.IDispatcherACCustomizable**.

   This class must implement the method in **IDispatcherACCustomizable**.

4. In the handler class, implement the **execute(ExecutionEvent)** method to call the **AdminDialogService.OpenAdminDialog()** method and pass an instance of the filter class created previously.

> Note:
>
> If you use this method, the Dispatcher request administration console ignores the filter class defined by extending the extension point. The Dispatcher request administration console only uses the filter class passed in the **OpenAdminDialog()** method in such a case.

10. Troubleshooting

Troubleshoot Dispatcher

Checklist to handle errors

- Ensure that the installation and configurations are properly done.
- Follow instructions in the documentation according to the order of dependencies.

  Example:

  To run the **ideastojt** translator, ensure that the **ideastojt** translator is installed according to the I-deas installation guide.

- Check whether all the property files are configured correctly.

  When you are not sure of the property values, try to be as close as possible to the default property values.

- Ensure that all prerequisite software is installed and tested before installing the dispatcher client.

  Also ensure the correct versions of the prerequisite software are installed.

- Restart the dispatcher client and rich client after any property changes.
- Ensure that service access rules are added.
- Ensure that you have permissions to write to the staging directory.
- Ensure that you have file storage space to run translation services.

Isolate translation problems

1. To isolate the application that is causing the translation failure, run the translator in different contexts:

   a. Run the translator in a standalone mode from the command line. For information about running the translator from the command line, see the translator documentation.

      It the translator fails, it is a translator issue. See the translator documentation for information about fixing it.

      If the translator works, go to the next step.

b. Perform a translation within the context of Dispatcher Server using the administration client.

   If the translation fails, it is a Dispatcher Server issue.

   If the translation works, go to the next step.

c. Perform a translation within Teamcenter.

   If the translation fails, it is a Teamcenter dispatcher client issue.

2. After the issue is identified as a Teamcenter dispatcher client issue, you can further isolate the problem by setting debug levels to higher values in the **log4j2.xml** file.

   More debug information is available in:

   - Console in which the dispatcher client started.
   - Service logs are available in the *Log-Directory***/Dispatcher/process/DispatcherClient.log** file.
   - Task logs are available in the *Log-Directory***/Dispatcher/***Task-ID*/*Task-ID***_ts.log** directory.

3. Delete or back up old logs to ensure that you are not looking at wrong files for debug messages.

4. Create a new query for **DispatcherRequest** using Query Builder. This is useful to check whether the **DispatcherRequest** object is created in the Teamcenter database.

5. Shut down the dispatcher client. Use the on-demand dispatcher client to trigger a translation request.

   a. Start the dispatcher client. Check the console, the **DispatcherClient.log** file, or task log files to see whether the dispatcher client was successful in extracting files from Teamcenter and uploading them to a staging location. Confirm that files are in a staging directory. If extraction does not occur for the translation request, query **DispatcherRequest** in the rich client to see whether a **DispatcherRequest** object is created.

   b. After extraction, the task is submitted to the scheduler. Check whether the task was successfully submitted to the scheduler and that you get back a translation successful event from Dispatcher Server. The Dispatcher Server events should appear in the following order:

      A. Submitted the task successfully

      B. Started translation

      C. Completed successfully

Confirm that result files (JT, CGM) are available on the file server staging directory. If translation fails, you get a translation error event. If translation fails, use input files in the file server staging directory to translate independent of dispatcher client using Dispatcher Server administration client or the core standalone translator.

If translation fails, the input files have some problem.

c. Check whether the dispatcher client was successful in downloading result files and attaching result files back to Teamcenter. If the download is successful and the attachment failed, check whether file mapping failed (file mapping fails if result files do not map one on one with source files). This can occur if translators are not configured per dispatcher client requirements.

Setting debug levels

For all Dispatcher services, the **LogVolumeLocation** property defines the location of the log files. This property can be found in the following locations:

- Module

  DISP_ROOT/**Module/conf/transmodule.properties**

- Scheduler

  DISP_ROOT/**Scheduler/conf/transscheduler.properties**

- Dispatcher client

  DISP_ROOT/**DispatcherClient/conf/DispatcherClient.properties**

You can define the debug level for logs in the **log4j2.xml** file. This file is located in the *DISP_ROOT*/*Dispatchert-component*/**conf** directory.

To change the debug level, change the value of the **level value** property to the debug level you want.

Information about the supported debug levels is as follows.

```
<!-- Supported Levels of logging are -->
<!-- "OFF"      - The OFF Level is intended to turn off logging. -->
<!-- "FATAL"    - The FATAL level designates very severe error
```

```
     events that will presumably lead the application to abort. --

<!-- "ERROR"    - The ERROR level designates error events that might
still
 allow the application to continue running. -->

<!-- "WARN"     - The WARN level designates potentially harmful
situations. -->

<!-- "INFO"     - The INFO level designates informational messages that

 highlight the progress of the application at coarse-grained level. -->

<!-- "DEBUG"    - The DEBUG Level designates fine-grained informational
events
that are most useful to debug an application. -->

<!-- "TRACE"    - The TRACE Level designates finer-grained informational
events
than the DEBUG and tracks the start and end of method calls. -->

<!-- "${DBGHG}" - The ${DBGHG} Level designates very fine-grained
information

 than TRACE. Used to debug code which has lots of output. -->

<!-- "ALL"      - The ALL Level is intended to turn on all logging. -->


<logger name="Process">

<level value="INFO"/>

<appender-ref ref="ProcessAppender"/>

<appender-ref ref="ProcessConsoleAppender"/>

</logger>


<logger name="Task">

<level value="INFO"/>

<appender-ref ref="TaskAppender"/>

<appender-ref ref="TaskConsoleAppender"/

</logger>
```

Query translation request objects

To check whether a translation request object is created in the database, create a new query in the Query Builder application as follows:

1. In the **Query Builder** pane, type **DispatcherRequest** in the **Name** box.

   This specifies the name for the query.

2. Click the **Search Class** button  , and select **DispatcherRequest** as the search class.

3. In the **Attribute Selection** pane, double-click the **State** attribute to add the **State** attribute to the **Search Criteria** table.

4. In the **Search Criteria** pane, set the **Default Value** attribute to *****.

5. Click the **Create** button to create the query definition.

Unable to run Dispatcher components as Windows service due to missing DLL file

You cannot run Dispatcher components as Windows service if the **msvcr71.dll** file is missing from your Windows system directory. To add this file to the Windows system directory:

- Copy the **msvcr71.dll** file from the *Teamcenter-install-kit***install\install\jre\bin** directory to your Windows system directory.

  For example, copy **msvcr71.dll** to **C:\Windows\SysWOW64**.

  OR

- Add the *Teamcenter-install-kit***install\install\jre\bin** directory to the system path variable.

Dispatcher requests go to terminal state due to large number of files open

On a Linux server, if the number of open dispatcher requests is greater than the number of files the server can handle, the extra dispatcher requests go to the terminal state.

To fix this, run the **ulimit** utility and update the **nofile** argument with the number of files or requests you want Dispatcher to handle at a time.

Example:

```
$ ulimit -n 65536
```

Support for Unicode and Cyrillic character encoding

Unicode character encoding

To support Unicode character encoding, add the following after the classpath statement in the *DISPATCHER_ROOT***DispatcherClient\bin\runDispatcherClient.bat** file (Windows example):

```
set JAVAARG=%JAVAARG% -Dfile.encoding=UTF-8
```

Cyrillic character encoding

To support Cyrillic character encoding, add the **cp866** character set for the following files:

1. Open the *DISPATCHER_ROOT***Scheduler\conf\log4j2.xml**, *DISPATCHER_ROOT***Module\conf\log4j2.xml**, and *DISPATCHER_ROOT***DispatcherClient\conf\log4j2.xml** files.

2. In **ProcessConsoleAppender** and **TaskConsoleAppender**:

   Change

   ```
   <PatternLayout pattern="%d %-5p - %m%n"/>
   ```

   To

   ```
   <PatternLayout charset="cp866" pattern="%d %-5p - %m%n"/>
   ```

3. Save all the files.

Troubleshoot Translators

NXToPvDirect translations hang

When translations for NxToPvDirect are submitted to the Dispatcher Server, the translator launches, but hangs and never completes the operation.

When this translator runs in batch mode it must connect and log on to Teamcenter as a validated user. When the dispatcher module launches the translator, the validated user logging on to Teamcenter must be the same user as the Module, meaning that the translator must be running as the same user name as the Module service. There are two solutions for this:

- Run the Module as a user with a valid Teamcenter user. This is the correct solution for this problem.

- Modify the launch script used by Dispatcher to start the translator to add the user name and password as a parameter to the UgToPvDirect translator. This is not the best solution because this exposes the

user name and password in a text file. If this approach is chosen, the Dispatcher batch script must be properly secured to avoid anyone (other than the module and translator) from being able to read or get access to this file.

Linux example:

```
# Using single part option with individual @DB strings
#
if [ $1 = -single_part ]; then
   if [ $3 = defaultdef ]; then
      export UGII_DEFAULTS_FILE=
   else
      export UGII_DEFAULTS_FILE=$3
   fi
   export UGII_LOAD_OPTIONS=
   $UGII_BASE_DIR/pvtrans/run_ugtopv -single_part
      -config=$UGII_CONFIG_FILE -noupdate -pim=yes $4 $5 $6 $7 $8 $9
#
# Copying latest UG release status.
#
elif [ $1 = copyugrelstatus ]; then
   . $TC_DATA/tc_profilevars
   $TC_ROOT/bin/tc_workflow_postprocess $2 $3 $4 $5 $6 $7 $8 $9
#
# Using text file with @DB Strings
#
else
   $UGII_BASE_DIR/pvtrans/run_ugtopv
   -config=$UGII_CONFIG_FILE -noupdate
   -pim=yes $3 $4 $5 $6 $7 $8 $9
fi
exitvalue=$?
if [[ `uname` = SunOS ]] then
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH_ORG
elif [[ `uname` = HP-UX ]] then
        export SHLIB_PATH=$SHLIB_PATH_ORG
elif [[ `uname` = AIX ]] then
        export LIBPATH=$LIBPATH_ORG
elif [[ `uname` = IRIX64 ]] then
        export LD_LIBRARY_PATH=$LD_LIBRARY_PATH_ORG
        export LD_LIBRARYN32_PATH=$LD_LIBRARYN32_PATH_ORG
fi
exit $exitvalue
```

Windows example:

```
REM
REM Using text file with @DB Strings
```

```
REM
"%UGII_BASE_DIR%\PVTRANS\run_ugtopv.bat"
       -config="%UGII_CONFIG_FILE%"
       -noupdate "-pim=yes" %3 %4 %5 %6 %7 %8 %9
goto done
REM
REM Using single part option with individual @DB strings
REM
:single_part
set UGII_DEFAULTS_FILE=%3
if %3==defaultdef set UGII_DEFAULTS_FILE=
set UGII_LOAD_OPTIONS=
"%UGII_BASE_DIR%\PVTRANS\run_ugtopv.bat"
       -single_part -config="%UGII_CONFIG_FILE%"
       -noupdate "-pim=yes" %4 %5 %6 %7 %8 %9
goto done
REM
REM Copying latest UG release status.
REM
:copyugrelstatus
call %TC_DATA%\tc_profilevars
"%TC_ROOT%\bin\tc_workflow_postprocess.exe" %2 %3 %4 %5 %6 %7 %8 %9
goto done
```

Multiple named references added after the translation process

When you run a translator, each time a named reference is added with an incremented number.

Example:

- Item revision: *XXXXXXXX/001*
- UGMASTER: *XXXXXXXX/001*
- CATPart: *XXXXXXXX/001*
- Direct model: *XXXXXXXX/001*
- Named reference: *JTPart Part$*.**jt**
- Named reference: *JTPart XXXXXX_001*.**jt**
- Named reference: *JTPart XXXXXX_002*.**jt**
- Named reference: *JTPart XXXXXX_003*.**jt**
- Named reference: *JTPart XXXXXX_004*.**jt**

To prevent the system from creating multiple named references of the dataset version, create the **TRANSLATION_PROXY_USER** user preference and set the value to *Dispatcher-client-proxy-user*.

View runtime status of Dispatcher services and Dispatcher client history

The **–ping** command is provided for all the three Dispatcher services. This command provides the runtime status of these services.

> Note:
>
> The ping command is performance intensive; use it only when required.

- Dispatcher scheduler: The **runscheduler.bat –ping** (Windows) or **runscheduler.sh –ping** (Linux) ping command provides runtime information on the scheduler and modules associated with the scheduler.
- Dispatcher module: The **runmodule.bat –ping** (Windows) or **runmodule.sh –ping** (Linux) ping command provides runtime information on the module.
- Dispatcher client: The **runDispatcherClient.bat –ping** (Windows) or **runDispatcherClient.sh –ping** (Linux) ping command provides runtime information on the Dispatcher client, scheduler, and the modules associated with the scheduler.

The **–history** command is provided for the Dispatcher client. It is used to parse the history log and provide a detailed summary of tasks performed per day. The **runDispatcherClient.bat –history** (Windows) or **runDispatcherClient.sh –history** (Linux) command provides information on the usage of the Dispatcher client.